Efficient Approximate Query Processing in Peer-to-Peer Networks

Benjamin Arai, *Student Member*, *IEEE*, Gautam Das, Dimitrios Gunopulos, *Member*, *IEEE*, and Vana Kalogeraki, *Member*, *IEEE*

Abstract—Peer-to-peer (P2P) databases are becoming prevalent on the Internet for distribution and sharing of documents, applications, and other digital media. The problem of answering large-scale ad hoc analysis queries, for example, aggregation queries, on these databases poses unique challenges. Exact solutions can be time consuming and difficult to implement, given the distributed and dynamic nature of P2P databases. In this paper, we present novel sampling-based techniques for approximate answering of ad hoc aggregation queries in such databases. Computing a high-quality random sample of the database efficiently in the P2P environment is complicated due to several factors: the data is distributed (usually in uneven quantities) across many peers, within each peer, the data is often highly correlated, and, moreover, even collecting a random sample of the peers is difficult to accomplish. To counter these problems, we have developed an adaptive two-phase sampling approach based on random walks of the P2P graph, as well as block-level sampling techniques. We present extensive experimental evaluations to demonstrate the feasibility of our proposed solution.

Index Terms—Approximation methods, computer networks, distributed databases, distributed database query processing, distributed estimation, database systems, distributed systems.

1 Introduction

1.1 Peer-to-Peer (P2P) Databases

The P2P network model is quickly becoming the preferred medium for file sharing and distributing data over the Internet. A P2P network consists of numerous peer nodes that share data and resources with other peers on an equal basis. Unlike traditional client-server models, no central coordination exists in a P2P system; thus, there is no central point of failure. P2P networks are scalable, fault tolerant, and dynamic, and nodes can join and depart the network with ease. The most compelling applications on P2P systems to date have been file sharing and retrieval. For example, P2P systems such as Napster [30], Gnutella [17], KaZaA [22], and Freenet [15] are principally known for their file sharing capabilities, for example, the sharing of songs, music, and so on. Furthermore, researchers have been interested in extending sophisticated infrared (IR) techniques such as keyword search and relevance retrieval to P2P databases.

1.2 Aggregation Queries

In this paper, however, we consider a problem on P2P systems that is different from the typical search and retrieval applications. As P2P systems mature beyond file sharing applications and start getting deployed in increasingly sophisticated e-business and scientific environments, the vast amount of data within P2P databases poses a different challenge that has not been adequately researched thus far, that is, how *aggregation queries* on such databases can be answered. Aggregation queries have the potential of finding applications in decision support, data analysis, and data mining. For example, millions of peers across the world may be cooperating on a grand experiment in astronomy, and astronomers may be interested in asking decision support queries that require the aggregation of vast amounts of data covering thousands of peers. In addition, there is real-world value for *aggregation queries* in network monitoring scenarios such as temperature and anomaly detection in sensor networks [39], Intrusion Detection Systems [26], [29], and application signature analysis [35] in P2P networks. Sensor networks can directly benefit from aggregation of traffic analysis data by offering a more efficient means of computing various network-based aggregates such as the average message size and maximum data throughput within the network, with minimal energy consumption and decreased response times.

We make the problem more precise as follows: Consider a single table T that is distributed over a P2P system; that is, the peers store horizontal partitions (of varying sizes) of this table. An aggregation query such as the following may be introduced at any peer (this peer is henceforth called the *query node*).

Aggregation query

SELECT Agg-Op(Col) FROM T WHERE selection-condition

In the above query, the *Agg-Op* may be any aggregation operator such as SUM, COUNT, AVG, and so on, *Col* may be any numeric measure column of T or even an expression involving multiple columns, and the *selection condition* decides which tuples should be involved in the aggregation. Although our main focus is on the above standard SQL aggregation operators, we also briefly discuss other interesting statistical estimators

- *B. Arai, D. Gunopulos, and V. Kalogeraki are with the Computer Science and Engineering Department, University of California, Riverside, Riverside, CA 92507. E-mail: {barai, dg, vana}@cs.ucr.edu.*
- *G. Das is with the Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX 76019. E-mail: gdas@cse.uta.edu.*

Manuscript received 28 Feb. 2006; revised 16 Nov. 2006; accepted 24 Jan. 2007; published online 5 Feb. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0105-0206.
Digital Object Identifier no. 10.1109/TKDE.2007.1064.

such as medians, quantiles, histograms, and distinct values.

Although aggregation queries have been heavily investigated in traditional databases, it is not clear that these techniques will easily adapt to the P2P domain. For example, decision support techniques such as online analytical processing (OLAP) commonly employ materialized views; however, the distribution and management of such views appear difficult in such a dynamic and decentralized domain [21], [13]. In contrast, the alternative of answering aggregation queries at runtime "from scratch" by crawling and scanning the entire P2P repository is prohibitively slow.

1.3 Approximate Query Processing (AQP)

Fortunately, it has been observed that in most typical data analysis and data mining applications, timeliness and interactivity are more important considerations than accuracy; thus, data analysts are often willing to overlook small inaccuracies in the answer, provided that the answer can be obtained fast enough. This observation has been the primary driving force behind the recent development of *AQP* techniques for aggregation queries in traditional databases and decision support systems [11], [3], [8], [10], [1], [16], [7], [9], [27]. Numerous AQP techniques have been developed: The most popular ones are based on random sampling, where a small random sample of the rows of the database is drawn, the query is executed on this small sample, and the results are extrapolated to the whole database. In addition to simplicity of implementation, random sampling has the compelling advantage that, in addition to an estimate of the aggregate, one can also provide confidence intervals of the error, with high probability. Broadly, two types of sampling-based approaches have been investigated: 1) *precomputed samples*, where a random sample is precomputed by scanning the database and the same sample is reused for several queries and 2) *online samples*, where the sample is drawn "on the fly" upon encountering a query.

1.4 Goal of the Paper

In this paper, we also approach the challenges of decision support and data analysis on P2P databases in the same manner; that is, we investigate what it takes to enable AQP techniques on such distributed databases.

Goal of the Paper: Approximating Aggregation Queries in P2P Networks.

*Given an aggregation query and a desired error bound at a query node peer, compute with "*minimum cost*" an approximate answer to this query that satisfied the error bound.*

The *cost* of query execution in traditional databases is usually a straightforward concept: It is either I/O cost or CPU cost or a combination of the two. In fact, most AQP approaches simplify this concept even further by just trying to minimize the number of tuples in the sample, thus making the assumption that the sample size is directly related to the cost of query execution. However, in P2P networks, the cost of query execution is a combination of several quantities such as the number of *participating peers*, the *bandwidth* consumed (that is, the amount of data shipped over the network), the number of *messages* exchanged, the *latency* (the time to propagate the query across multiple peers and receive replies), the *I/O cost* of accessing data from participating peers, the *CPU cost* of processing data at participating peers, and so on. In this paper, we shall be concerned with *latency* (the time to propagate the query across multiple peers and receive replies) as our primary quantity to minimize though our technique could be easily extended to deal with other cost metrics.

1.5 Challenges

Let us now discuss what it takes for sampling-based AQP techniques to be incorporated into P2P systems. We first observe that two main approaches have emerged for constructing P2P networks today: *structured* and *unstructured*. Structured P2P networks (such as Pastry [33] and Chord [37]) are organized in such a way that data items are located at specific nodes in the network, and nodes maintain some state information to enable efficient retrieval of the data. This organization maps data items to particular nodes and assumes that all nodes are equal in terms of resources, which can lead to bottlenecks and hot spots. Our work focuses on unstructured P2P networks, which makes no assumption about the location of the data items in the node, and nodes are able to join the system at random times and depart without a priori notification. Several recent efforts have demonstrated that unstructured P2P networks can be used efficiently for multicast distributed object location and information retrieval [12], [27], [38].

For AQP in unstructured P2P systems, attempting to adapt the approach of precomputed samples is impractical for several reasons: 1) It involves scanning the entire P2P repository, which is difficult, 2) since no centralized storage exists, it is not clear where the precomputed sample should reside, and 3) the very dynamic nature of P2P systems indicates that precomputed samples will quickly become stale, unless they are frequently refreshed.

Thus, the approach taken in this paper is to investigate the feasibility of online sampling techniques for AQP on P2P databases. However, online sampling approaches in P2P databases pose their own set of challenges. To illustrate these challenges, consider the problem of attempting to draw a *uniform random sample* of n tuples from such a P2P database containing a total of N tuples. To ensure a true uniform random sample, our sampling procedure should be such that each subset of n tuples out of N should be equally likely to be drawn. However, this is an extremely challenging problem due to two reasons:

- Picking even a set of uniform random peers is a difficult problem, as the query node does not have the Internet Protocol (IP) addresses of all peers in the network. This is a well-known problem that other researchers have tackled (in different contexts) by using *random-walk* techniques on the P2P graph [16], [24], [4]. That is, where a Markovian random walk is initiated from the query node that picks adjacent peers to visit, with equal probability and under certain connectivity properties, the random walk is expected to rapidly reach a stationary distribution. If the graph is badly clustered with small cuts, then

this affects the speed at which the walk converges. Moreover, even after convergence, the stationary distribution is *not uniform*; in fact, it is skewed toward giving higher probabilities to nodes with larger degrees in the P2P graph.

- Even if we could select a peer (or a set of peers) uniformly at random, it does not make the problem of selecting a *uniform random set of tuples* much easier. This is because visiting a peer at random has an associated overhead; thus, it makes sense to select multiple tuples at random from this peer during the same visit. However, this may compromise the quality of the final set of tuples retrieved, as the tuples within the same peer are likely to be *correlated*. For example, if the P2P database contained listings of, say, movies, then the movies stored on a specific peer are likely to be of the same genre. This correlation can be reduced if we select just one tuple at random from a randomly selected peer; however, the overheads associated with such a scheme will be intolerable.

1.6 Our Approach

We briefly describe the framework of our approach. Essentially, we abandon trying to pick true uniform random samples of the tuples, as such samples are likely to be extremely impractical to obtain. Instead, we consider an approach where we are willing to work with *skewed samples*, provided that we can accurately estimate the skew during the sampling process. To get the accuracy in the query answer desired by the user, our skewed samples can be larger than the size of a corresponding uniform random sample that delivers the same accuracy; however, our samples are much more cost efficient to generate.

Although we do not advocate any significant preprocessing, we assume that certain aspects of the P2P graph are known to all peers, such as the average degree of the nodes, a good estimate of the number of peers in the system, certain topological characteristics of the graph structure, and so on. Estimating these parameters via preprocessing are interesting problems in their own right; however, we omit these details from this paper. The main point that we make is that these parameters are relatively slow to change and thus do not have to be estimated at query time: It is the data contents of peers that changes more rapidly; hence, the random sampling process that picks a representative sample of tuples has to be done at runtime.

Our approach has two major phases. In the first phase, we initiate a fixed-length random walk from the query node. This random walk should be long enough to ensure that the visited peers represent a close sample from the underlying stationary distribution (the appropriate length of such a walk is determined in a preprocessing step). We then retrieve certain information from the visited peers, such as the number of tuples, the aggregate of tuples (for example, SUM, COUNT, AVG, and so forth) that satisfy the selection condition, and send this information back to the query node. This information is then analyzed at the query node to determine the skewed nature of the data that is distributed across the network, such as the variance of the aggregates of the data at peers, the amount of correlation between tuples that exists within the same peers, the variance in the degrees of individual nodes in the P2P graph (recall that the degree has a bearing on the probability that a node will be sampled by the random walk), and so on. Once this data has been analyzed at the query node, an estimation is made on how much more samples are required (and in what way should these samples be collected) so that the original query can be optimally answered within the desired accuracy, with high probability. For example, the first phase may recommend that the best way to answer this query is to visit m' more peers and, from each peer, randomly sample t tuples. We mention that the first phase is not overly driven by heuristics. Instead, it is based on underlying theoretical principles such as the theory of random walks [16], [24], [4] as well as statistical techniques such as cluster sampling, block-level sampling, and cross validation [11], [18].

The second phase is then straightforward: A random walk is reinitiated, and tuples are collected according to the recommendations made by the first phase. Effectively, the first phase is used to "sniff" the network and determine an optimal-cost "query plan," which is then implemented in the second phase. For certain aggregates such as COUNT and SUM, further optimizations may be achieved by pushing the selections and aggregations to the peers; that is, the local aggregates instead of raw samples are returned to the query node, which are then composed into a final answer.

In addition, we explore in-network techniques for dissemination of values throughout the network. We accomplish this through a hybrid technique building upon the Gossip protocol. A Gossip protocol is executed in rounds. For each round, participating peers select adjacent peers uniformly at random sharing information. The *Gossip* protocol exploits a communication mechanism where peers diffuse local aggregates with adjacent peers. This process relies heavily upon *mass conversation*, which describes that the average of all of the sums of individual peers is the correct average, and the sum of all of the weights is n [23]. In general, as the number of passes of the Gossip protocol increases, values of participating peers are increasingly diffused through the network (in our case, the local groups); therefore, sampling-diffused values provide a better representation of the values contained in the network as opposed to a single peer.

The contributions of this paper are summarized as follows:

- We introduce the important problem of AQP in P2P databases, which is likely to be of increasing significance in the future.
- The problem is analyzed in detail, and its unique challenges are comprehensively discussed.
- Hybrid sampling technique maximizes per-peer in-network computation building upon the *Gossip* protocol.
- Adaptive two-phase sampling-based approaches are proposed based on well-founded theoretical principles.

- We present an adaptive approach for computing aggregates such as *COUNT*, *SUM*, *AVERAGE*, and *MEDIAN*.
- The results of extensive experiments are presented, which demonstrate the importance of the problem and the validity of our approaches.

The rest of this paper is organized as follows: In Section 2, we describe related work. We provide the foundation of our approach in Section 3, the algorithm in Section 4, and the hybrid solution to random sampling in Section 5. In Section 6, we present the experimental results, and we conclude in Section 7.

2 Related Work

P2P systems are becoming very popular because they provide an efficient mechanism for building large scalable systems [28]. Most recent work has focused on Distributed Hash Tables (DHTs) [32], [33], [37]. Such techniques provide scalability advantages over unstructured systems (such as Gnutella); however, they are not flexible enough for some applications, especially when nodes join or leave the network frequently or change their connections often.

Recent work has proposed different techniques for exact query processing in P2P systems. Most proposals use structured overlay networks (DHTs), such as CAN, Pastry, and Chord. Such techniques include PIER [19], DIM [27], or Pastry [33], and since they use DHTs, they have a different focus and are not directly applicable to our case. A hybrid system, Mercury [4], using routing hubs to answer range queries was also recently proposed. This system is also designed to provide exact answers to range queries. Exact solutions to OLAP queries have been considered in [13] and [21].

Methods to sample random peers in P2P networks have been proposed in [16], [24], and [4]. These techniques use Markov-chain random walks to select random peers from the network. Their results show that when certain structural properties of the graph are known or can be estimated (such as the second eigenvalue of the graph), the parameters of the walk can be set so that a representative sample of the stationary distribution can be collected with high probability. In [4], it is shown that if the graph is an expander, then a random walk converges to the stationary distribution in $\mathrm{O}(\log \mathrm{M})$ steps, where M is the number of peers in the network.

There are known techniques for computing approximate aggregates in distributed settings (most notably, the *Gossip* protocol [5], [6], [23]). The technique works generally as a preprocessing step where all peers in a network attempt to mix data among adjacent peers, eventually converging upon a single value. The inability to contact all nodes in the network makes it exceedingly difficult to *Gossip* in the traditional sense.

Our work also generalizes to the P2P domain and previous work on AQP in relational databases. Recent work in [11], [3], [8], [10], [1], [16], [7], [9], and [27] has developed powerful techniques for employing sampling in the database engine to approximate aggregation queries and to estimate database statistics. Recent techniques have focused on providing formal foundations and algorithms for block-level sampling and are thus most relevant to our work. The objective in block-level sampling is to derive a representative sample by only randomly selecting a set of disk blocks of a relation [11], [18]. Specifically, [11] presents a technique for histogram estimation, which uses cross validation to identify the amount of sampling required for a desired accuracy level. In addition, [18] considers the problem of deciding what percentage of a disk block should be included in the sample, given a cost model.

3 Foundations of Our Approach

In this section, we discuss the principles behind our approach for AQP on P2P databases. Our actual algorithm is described in Section 4.

3.1 The Peer-to-Peer Model

We assume an unstructured P2P network represented as a graph $G = (P, E)$, with a vertex set $P = \{p_1, p_2, \ldots, p_M\}$ and an edge set E. The vertices in P represent the peers in the network, and the edges in E represent the connections between the vertices in P. Each peer p is identified by the processor's IP address and a port number ($\mathrm{IP_p}$ and $\mathrm{port_p}$). The peer p is also characterized by the capabilities of the processor on which it is located, including its CPU speed $\mathrm{p_{cpu}}$, memory bandwidth $\mathrm{p_{mem}}$, and disk space $\mathrm{p_{disk}}$. The node also has a limited amount of bandwidth to the network, noted by $\mathrm{p_{band}}$. In unstructured P2P networks, a node becomes a member of the network by establishing a connection with at least one peer currently in the network. Each node maintains a small number of connections with its peers: The number of connections is typically limited by the resources at the peer. We denote the number of connections that a peer is maintaining by $\mathrm{p_{conn}}$.

The peers in the network use the Gnutella P2P protocol to communicate. The Gnutella P2P protocol supports four message types (Ping, Pong, Query, and Query_Hit), of which the Ping and Pong messages are used to establish connections with other peers, and the Query and Query_Hit messages are used to search in the P2P network. Gnutella, however, uses a naive Breadth-First Search (BFS) technique in which queries are propagated to all the peers in the network and thus consumes excessive network and processing resources and results in poor performance. Our approach, on the other hand, uses a probabilistic search algorithm based on random walks. The key idea is that each node forwards a query message, called *walker*, randomly to one of its adjacent peers. This technique is shown to improve the search efficiency and reduce unnecessary traffic in the P2P network.

3.2 Query Cost Measures

As mentioned in Section 1, the cost of the execution of a query in P2P databases is more complicated than equivalent cost measures in traditional databases. The primary cost measure that we consider is *latency*, which is the time that it takes to propagate the query across multiple peers and receive replies at the query node. In our algorithm, latency can be approximated by the number of peers that participate in the random walk. This measure is appropriate for our algorithm because it performs a single random walk starting from the query node. Thus, latency becomes proportional to the total number of visited peers in the random walk.

To see this, we note that the aggregation operator (as well as the selection filter and IP address of the query node) can be pushed to each visited peer. Once a peer is visited by the algorithm, the peer can be instructed to simply execute

the original query on its local data and send only the aggregate and the degree of the node back to the query node, from which the query node can reconstruct the overall answer. Moreover, this information can be sent directly without necessitating any intermediate hops, as the visited peer knows the IP address of the query node from which the query originated. This is reasonable, considering that the IP address can be pushed to visited peers along with the aggregation operator and the P2P networks such as Kazaa run on top of a TCP/IP layer, making it feasible to make direct connections with peers. Thus, the bandwidth requirement of such an approach is uniformly very small for all visited peers: They are not required to send more voluminous raw data (for example, all or parts of the local database) back to the query node.

In approximating latency by the number of peers participating in the random walk, we also make the implicit assumption that the overhead of visiting peers dominates the costs of local computations (such as execution of the original query on the local database). This is, of course, true if the local databases are fairly small. To ensure that the local computations remain small even if local databases are large, our approach in such cases is to execute the aggregation query only on a small fixed-sized random sample of the local data (that is, we *subsample* from the peer), scale the result to the entire local database, and send the scaled aggregate back to the query node. This way, we ensure that the local computations are uniformly small across all visited peers.

In contrast, suppose that instead of a fixed sized sample, we decided on sampling a *fixed fraction* of a visited peer's local database. The main problem with this approach is that it complicates the query cost model. Now, local processing costs cannot be ignored and, thus, latency cost of executing a query cannot be modeled as simply being proportional to the number of visited peers (or even the overall number of sampled tuples). The latency now becomes a complex (and, perhaps, system dependent) function of the cost of visiting peers and local query processing costs. The consequence of a complicated latency model is that it now becomes difficult to have a principled two-phase approach to solving the problem because the first phase now has the task of determining how many peers should be sampled in the second phase so that the target accuracy can be achieved with minimum latency. Moreover, even if the first phase can somehow determine the number of peers to visit in the second phase, the actual latency cost of the second phase is unpredictable. It depends on the type of peers we visit, as peers with large databases will increase latency, whereas peers with small databases will decrease latency.

In summary, for SUM and COUNT aggregates, latency is shown to be proportional to the number of peers participating in the random walk. Thus, our goal is to minimize the number of peers that must be visited in order to arrive at an approximate answer with the desired accuracy.

3.3 Random Walk in Graphs

In seeking a random sample of the P2P database, we have to overcome the subproblem of how a random sample of the peers themselves can be collected. Unrepresentative samples of peers can quickly skew results, producing erroneous aggregation statistics. Sampling in a nonhierarchical decentralized P2P network presents several obstacles in obtaining near-uniform random samples. This is because no peer (including the query node) knows the IP addresses of all other peers in the network: They are only aware of their immediate neighbors. If this were not the case, then, clearly, the query node could locally generate a random subset of IP addresses from among all the IP addresses and visit the appropriate peers directly. We note that this problem is not encountered in traditional databases, as even if one has to resort to cluster (or block-level) sampling such as in [11] and [18], obtaining an efficient sample of the blocks themselves is straightforward.

This problem has been recognized in other contexts (see [16] and the references therein), and interesting solutions based on *Markov-chain random walks* have been proposed. We briefly review such approaches here. A Markov-chain random walk is a procedure that is initiated at the query node, and for each visited peer, the next peer to visit is selected with equal probability from among its neighbors (and itself and, thus, self loops are allowed). It is well known that if this walk is carried out long enough, then the eventual probability of reaching any peer p will reach a stationary distribution. To make this more precise, let $P = \{p_1, p_2, \ldots, p_{\mathrm{M}}\}$ be the entire set of peers, let E be the entire set of edges, and let the degree of a peer p be $\deg(p)$. Then, the probability of any peer p in the stationary distribution is

$$prob(p) = \frac{\deg(p)}{2|E|}.$$

It is important to note that the above distribution is *not uniform*: The probability of each peer is proportional to its degree. Thus, even if we can efficiently achieve this distribution, we will have to compensate for the fact that the distribution is skewed as above if we have to use samples drawn from it for answering aggregation queries.

The main issue that has concerned researchers has been the *speed of convergence*.[1] Most results have pointed to certain broad connectivity properties that the graph should possess for this to happen. In particular, it has been shown that if the transition probabilities that govern the random walk on the P2P graph are modeled as an $M \times M$ matrix, then the *second eigenvalue*[2] plays an important role in these convergence results. The second eigenvalue describes connectivity properties of graphs, in particular, whether the graph has a small *cut size*,[3] which would adversely impact the length of the walk necessary to arrive at convergence.

As the results in [16] show, if the P2P graph is well connected (that is, it has a small second eigenvalue, and a minimum degree of the graph is large), then the random walk quickly converges as it "loses memory" rapidly. In fact, under certain specific conditions of connectedness (for example, expander graphs that are common in P2P networks), convergence can be achieved in $\mathrm{O}(\log M)$ steps.

In our case, recall from Section 1 that we assume that we are allowed a certain amount of preprocessing to determine various properties of the P2P graph that will be useful at query time (under the assumption that the graph topology changes less rapidly compared to the data content at the peers). The speed of convergence of a random walk in this graph is determined in this preprocessing step, in addition to other useful properties such as the number of nodes M, the number of edges $|E|$, and so on. With respect to speed of

1. We define *speed of convergence* as how many hops h are necessary before one gets close to the stationary distribution.

2. The *second eigenvalue* tells how well the peers within the network are connected, that is, expander versus clustered sets of peers.

3. Given a partition of peers in two sets A and B, any edge crossing from A to be B is crossing the cut. The cut size is sum of the edges crossing A and B.

convergence, we essentially determine a *jump* parameter j that determines how many peers can be skipped between selections of peers for the sample. As the jump increases, the correlation between successive peers that are selected for the sample decreases rapidly.

3.4 Sampling Theorems

In this section, we shall develop the formal sampling theorems that drive our algorithm. We shall show how the tuples that are retrieved from the first phase of our algorithm can be utilized to recommend how the second phase should be executed, that is, the "query plan" for answering the query approximately so that a desired error is achieved.

We focus here on the COUNT aggregate for the purpose of illustrating our main ideas (our formal results can be easily extended for the SUM and AVERAGE cases). Finally, to keep the discussion simple, we assume that all local databases at peers are small, that is, subsampling is not required (our results can be extended for the subsampling case and, in fact, our algorithm in Section 4 does not make this assumption).

As discussed earlier, our algorithm has two phases. In the first phase, our algorithm will visit a predefined number of peers m by using a random walk such that the sample of visited peers will appear as if they have been drawn from the stationary distribution of the graph. The query will be executed locally at each visited peer, and the aggregates will be sent back to the query node, along with other information such as the degrees of the visited peers (from which information such as the peers' probabilities in the stationary distribution can be computed). The query node analyzes this information and then determines how many more peers need to be visited in the second phase. The theorems that we develop next provide the foundations on which the decisions in the first phase are made.

Recall that $P = \{p_1, p_2, \ldots, p_M\}$ is the set of peers.

For a tuple u, let $y(u) = 1$ if u satisfies the selection condition, and $y(u) = 0$ otherwise.

Let the aggregate for a peer p be $y(p) = \sum_{u \in p} y(u)$.

Let y be the exact answer for the query, that is, $y = \sum_{p \in P} y(p)$.

The query also comes with a desired error threshold Δ_{req}. The implication of this requirement is that if y' is the estimated count by our algorithm, then

$$|y - y'| \leq \Delta_{req}.$$

Now, consider a fixed-size sample of peers

$$S = \{s_1, s_2 \ldots s_{\mathrm{m}}\},$$

where each s_{i} is from P. This sample is picked by the random walk in the first phase. We can approximate this process as that of picking peers in m rounds, where in each round, a random peer s_{i} is picked from P, with probability $prob(s_{\mathrm{i}})$. We also assume that peers may be picked *with replacement*; that is, multiple copies of the same peer may be added to the sample, as this greatly simplifies the statistical derivations below.

Consider the quantity y'' defined as follows:

$$y'' = \frac{\sum_{s \in S} \frac{y(s)}{prob(s)}}{m}. \quad (1)$$

Theorem 1. *$E[y''] = y$, that is, y'', is an unbiased estimator of y.*

Proof. Intuitively, each sampled peer s tries to estimate y as $y(s)/prob(s)$, that is, by scaling its own aggregate by the inverse of its probability of getting picked. The final estimate y'' is simply the average of the m individual estimates.

To proceed with the proof, consider the simple case of only one sampled peer, that is, $m = 1$. In this case,

$$E[y''] = \sum_{p \in P} \left(\frac{y(p)}{prob(p)} \right) prob(p) = y.$$

To extend to any m, we make use of the *linearity of expectation* formula $E[X + Y] = E[X] + E[Y]$ for random variables X and Y (that need not even be independent). Thus, if the expected estimate of any single random peer is y, then the expected average estimate by m random peers is also y. □

We next need to determine the variance of the random variable y''.

Theorem 2 (Standard Error Theorem).

$$Var[y''] = \frac{\sum_{p \in P} \left(\frac{y(p)}{prob(p)} - y \right)^2 prob(p)}{m}.$$

Proof. To easily derive this variance, let us consider the simple case of only one sampled peer, that is, $m = 1$. In this case, it is easy to see that the variance is defined by the quantity:

$$C = \sum_{p \in P} \left(\frac{y(p)}{prob(p)} - y \right)^2 prob(p).$$

To extend to any m, we make use of the following formulas for variance: 1) $Var[aX] = a^2 Var[X]$ and 2) $Var[X + Y] = Var[X] + Var[Y]$, where X and Y are independent random variables, and a is a constant. By using these formulas, we can easily show that $Var[y''] = C/m$. □

The above **Standard Error Theorem** shows that the variance varies inversely as the sample size. The quantity C also represents the "badness" of the clustering of the data in the peers: The larger the C, the more the correlation among the tuples within peers and, consequently, the more peers need to be sampled to keep the variance of the estimator y'' small. Notice also that if we divide the variance by N^2, then we will effectively get the square of the error of the relative count aggregate if y'' was used as an estimator for y.

Our case is actually the reverse; that is, we are given a desired error threshold Δ_{req}, and the task is to determine the appropriate number of peers to sample that will satisfy this threshold. Of course, we have used a fixed-sized m in the first phase, so unless we are simply lucky, it is unlikely that this particular m will satisfy the desired accuracy. However, we can use the first phase more carefully to determine the appropriate sample size to draw in the second phase, say, m'.

The main task is to use the sample drawn in the first phase to try to estimate C because once we estimate C, we can determine m' by using Theorem 2. We suggest a simple *cross-validation* procedure as described below to estimate C

(this procedure is inspired by previous work in a different context; see [11]).

Consider two random samples of peers of size m, each drawn from the stationary distribution. Let y_1'' and y_2'' be the two estimates of y by these samples, respectively, according to (1). We define the *cross-validation* error (CVError) as $CVError = |y_1'' - y_2''|$.

Theorem 3.

$$E[CVError^2] = 2E[(y'' - y)^2].$$

Proof.

$$E[CVError^2] = E\Big[\big(y_1'' - y_2''\big)^2\Big] = E\Big[\big(y_1'' - y\big)^2\Big] + E\Big[\big(y_2'' - y\big)^2\Big] = 2E\Big[\big(y'' - y\big)^2\Big].$$

□

This theorem says that the expected value of the square of the CVError is two times the expected value of the square of the actual error.

This CVError can be estimated in the first phase by the following procedure. Randomly divide the m samples into halves and compute the CVError (for sample size $m/2$). We can then determine C by fitting this computed error and the sample size $m/2$ into the equation in Theorem 2. To get a somewhat more robust estimation for C, we can repeat the random halving of the sample collected in the first phase several times and take the average value of C. We also note that since the CVError is larger than the true error, the value of C is conservatively overestimated.

Once C is determined (that is, the "badness" of the clustering of data in the peers), we can determine the right number of peers to sample in the second phase m' to achieve the desired accuracy.

4 Our Algorithm

In this section, we present details of our two-phase algorithm for approximate answering of aggregate queries. For illustration, we focus on approximating COUNT queries (it can be easily extended to SUM, AVERAGE, and MEDIAN queries). The pseudocode of the algorithm is presented below.

4.1 COUNT

Our approach in the first phase is broken up into the following main components. First, we perform a random walk on the P2P network, attempting to avoid skewing due to graph clustering and vertices of high degree. Our walk skips j nodes between each selection to reduce the dependency between consecutive selected peers. As the jump size increases, our method increases overall bandwidth requirements within the database, but for most cases, small jump sizes suffice for obtaining random samples.

Second, we compute aggregates of the data at the peers and send these back to the query node.

Note that in Section 3, we had not formally discussed the issue of subsampling at peers: This was primarily done to keep the previous discussion simple. In reality, the local databases at some peers can be quite large, and aggregating them in their entirety may not be negligible as compared to the overhead of visiting the peer. In other words, the simplistic cost model of only counting the number of visited peers is inappropriate. In such cases, it is preferable to randomly subsample a small portion of the local database and apply the aggregation only to this subsample. Thus, the ideal approach for this problem is to develop a cost model that takes into account cost of visiting peers, as well as local processing costs. Moreover, for such cost models, an ideal two-phase algorithm should determine various parameters in the first phase, such as how many peers should be visited in the second phase and how many tuples should be subsampled from each visited peer. In this paper, we have taken a somewhat simpler approach in which we fix a constant t (determined at preprocessing time via experiments) such that if a peer has at most t tuples, then its database is aggregated in its entirety, whereas if the peer has more than t tuples, then t tuples are randomly selected and aggregated. Subsampling can be more efficient than scanning the entire local database, for example, by *block-level sampling*, in which only a small number of disk blocks are retrieved. If the data in the disk blocks are highly correlated, then it will simply mean that the number of peers to be visited will increase, as determined by our cross-validation approach at query time.

Third, we estimate the CVError of the collected sample and use that to estimate the additional number of peers that need to be visited in the second phase. For improving robustness, steps 2-4 in the cross-validation procedure can be repeated a few times, as well as the average squared CVError computed.

Once the first phase has completed, the second phase is then straightforward. We simply initiate a second random walk based on the recommendations of the first phase and compute the final aggregate.

4.2 SUM and AVERAGE

Although the algorithm has been presented for COUNT queries, it can be easily extended to other aggregates such as the SUM and AVERAGE by modifying the $y(Curr)$ value specified on line 8, phase 1 of the algorithm. For the SUM, no changes are required, and for the AVERAGE, (#tuples/#processTuples) is removed from $y(Curr)$, since no scaling is required.

4.3 MEDIAN

For more complex aggregates such as estimation of medians, quantiles, and distinct values, more sophisticated algorithms are required. In addition to computing COUNT, SUM, and AVERAGE aggregates, we can also efficiently estimate more difficult aggregates such as the MEDIAN. We propose an algorithm for computing the MEDIAN in a distributed fashion based upon comparing the rank distances of medians of individual peers. Our algorithm for computing the MEDIAN is given as follows:

1. Select m peers at random by using random walk.
2. Each peer s_j computes its median med_j and sends it to the query node, along with $prob(s_j)$.
3. The query node randomly partitions the m medians into two groups of $m/2$ medians: *Group1* and *Group2*.
4. Let med_{g1} be the weighted median of *Group1*, that is, such that the following is minimized:

$$abs\left(\sum_{\substack{med_j \in Group1,\\ med_j < med_{g1}}} 1/prob(s_j) - \sum_{\substack{med_j \in Group1,\\ med_j > med_{g1}}} 1/prob(s_j)\right).$$

5. Find the error between the median of *Group2* (say, med_{g2}) and the weighted rank of med_{g1} in *Group2*. That is, let

$$c = abs\left(\sum_{\substack{med_j \in Group2,\\ med_j < med_{g1}}} 1/prob(s_j) - \sum_{\substack{med_j \in Group2,\\ med_j < med_{g2}}} 1/prob(s_j)\right)/(m/2).$$

6. Select additional c^2/Δ_{req}^2 peers by using random walk.
7. Find and return the weighted median of the medians of the additional peers.

Similar to our COUNT algorithm, our technique for computing the MEDIAN offers great advantages over the naive approach of estimating the MEDIAN at the query node. By computing the MEDIAN at individual peers and sending these aggregates back to the query node, we reduce the number of messages sent over the network. This method can easily be extended to other aggregates such as the quantiles.

5 Hybrid Solution

In order to further improve the quality of our random sampling process, we have employed a hybrid sampling technique by allowing individually selected peers to perform additional sampling in parallel with the random sampling phase. We exploit the fact that during a random walk, previously selected peers can perform further independent processing while waiting for the final peer to be selected for sampling during the random-walk phase. The ability to execute in-network computation is a valuable tool for maximizing sample quality and reducing the required *jump size* for individual queries. Our hybrid technique can be utilized for many aggregate types including SUM, AVERAGE, COUNT, and MEDIAN queries.

Algorithm: COUNT queries

Predefined values
M : total number of peers in network
E : total number of edges in network
m : number of peers to visit in phase 1
j : jump size for random walk
t : max #tuples to be subsampled per peer

Inputs
Q : COUNT query with selection condition
$Sink$: peer where query is initiated
Δ_{req} : desired max error

Phase 1
// *Perform random walk*
1. $Curr = Sink$; $Hops = 1$;
2. while $(Hops < j * m)$ {
3. if $(Hops \,\%\, j)$
4. $Visit(Curr)$;
5. $Hops++$;
6. $Curr =$ random adjacent peer
7. }

// *Visit peer*
1. $Visit(Curr)$ {
2. if (#tuples of $Curr$) $<= t$) {
3. Execute Q on all tuples
4. else
5. Execute Q on t randomly sampled
6. tuples
7. }
8. $y(Curr) = \left(\frac{\#tuples}{\#processedTuples}\right) * (result\_of\_Q)$
10. Return $(y(Curr), deg(Curr))$ to $Sink$
11. }

// *Cross validate at sink*
1. Let $S = \{s_1, s_2, \ldots, s_m\}$ be the visited peers
2. Partition S randomly into halves: S_1 and S_2
3. Compute

$$y_1'' = \frac{\sum_{s \in S_1} y(s)/prob(s)}{m/2} \qquad y_2'' = \frac{\sum_{s \in S_2} y(s)/prob(s)}{m/2}$$

where $prob(s) = \frac{\deg(s)}{2E}$
4. Compute $CVError = |y_1'' - y_2''|$

5. Return $m' = (m/2) * \left(\frac{CVError^2}{\Delta_{req}^2}\right)$

Phase 2
1. Visit m' peers by using random walk
2. Let $S' = \{s_1, s_2, \ldots, s_m'\}$ be the visited peers
3. Return $y' = \frac{\sum_{s \in S'} y(s)/prob(s)}{m'}$

5.1 Hybrid Algorithm

Since each peer is limited to knowledge of adjacent peers, computing aggregates based upon a single start location (query node) limits the total number of peers available for processing. Each peer accessed is aware of the peers that make up the path from the query node to itself. Most queries are unable to reach all peers in a network either due to high per-message cost or query execution time requirements. Techniques exist for generating expander P2P topologies [31]: A fully decentralized approach for computing random samples is impractical. We address the possibility of highly connected networks by randomly selecting adjacent peers as opposed to flooding. Since we cannot fully exploit a decentralized approach for query processing, we propose a hybrid solution for random sampling, focusing on extending our technique with a hybrid in-network decentralized approach.

Upon selection of a peer p_i by the random-walk phase, p_i contains a *period* p_{i_period} where further processing may be performed to improve the quality of a peer's local data. The *period* p_{i_period} is defined as the number of hops remaining in the random-walk phase before the final peer p_m is selected for sampling. In order to exploit these *periods*, we propose an incremental decentralized sampling technique building upon the *Gossip* protocol [23]. The number of messages sent

over the network due to gossiping may be varied based upon the user-defined parameters r_a and r_r. Parameter r_a is the number of edges that a peer may randomly select for gossiping, and r_r is the maximum number of hops from p_i that gossiping is permitted. Regardless of the value of r_a or r_r, the number of messages sent to the query node remains constant. These two parameters combined allow the user to leverage in-network computation, without affecting the number of messages sent back to the query node, avoiding possible bottlenecks.

The *Gossip* protocol works as follows: For each peer in the path p_i, compute the sum of all of the tuples contained in the peer. For each randomly selected adjacent peer p_{ij} of p_i, send the aggregate values s_{ij} and an associated weight w_{ij} representing the contributions of the randomly selected adjacent peers of p_i. For each selected peer p_i, local sets are created $s_{i1\ldots n}$ and $w_{i1\ldots n}$ from contributed s_{ij} and w_{ij} values. From the adjacent peers, these sets are used to revaluate s_i and w_i for the current peer p_i. This process allows adjacent peers to mix data and allows local data sets to take into consideration both the local and the contributed values. Each round of the *Gossip* algorithm involves a single pass where neighboring peers contribute local values to adjacent peers. A single iteration of the *Gossip* algorithm executes as follows:

1. Each peer p_i maintains a local sum $\mathrm{s_i}$ and weight w_i. All weights w_i are initialized to 1 for average and 0 for sum.
2. For each peer, let $s_{i1\ldots n}$ represent the contributed sums from all randomly selected adjacent peers and $w_{i1\ldots n}$ represent the number of tuples contributed from each respective adjacent peer.
3. Each peer p_i randomly selects r_a adjacent peer's $p_{i1\ldots ra}$ for gossiping.
4. Assuming that the associated weights for each item in $s_{i1\ldots n}$ is 1, our new s_i value is the sum of the vector $s_{i1\ldots ra}$, and, similarly, w_i is the sum of the vector $w_{i1\ldots ra}$:

$$s_i = \sum_{j=0}^{m} s_{ij}, \qquad w_i = \sum_{j=0}^{m} w_{ij}.$$

5. Send $(1/2)s_i$ and $(1/2)w_i$ to randomly selected adjacent peers and p_i. By sharing (1/2), peers may exchange information between peers while still obeying the mass conservation requirement.
6. With each aggregate replaced with computed s_i and w_i, the new estimated aggregate is s_i/w_i.

Mixing between peers increases the diffusion of values through the network. For our purpose, there is no specific constraint on the number of iterations required before exiting. Under our hybrid approach, the algorithm attempts to maximize the amount of mixing per peer p_i by exploiting the *period* before a peer must send a sample back to the query node. As stated in [23], the diffusion speed of the network can be represented as $\mathrm{T}(n, \varepsilon) = O(\log n + \log 1/\varepsilon)$ for expander-type networks. In addition, for very long walks, convergence may occur before the final peer has been selected, but we can continue to perform gossiping, without loss of benefit, since P2P networks such as Gnutella [17] are, by nature, transient. Where peers are continually entering and leaving the network, gossiping can continue to diffuse new values as peers enter the network.

Simply, since we know how many peers remain to be selected by the random-walk phase, the lower bound for the

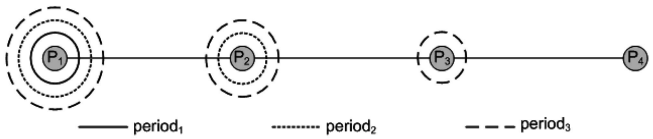

Fig. 1. Each ring represents the increase in gossiping per period for each peer.

period p_{i_period} is the remaining number of hops required to obtain the required sample, given the specified *jump size* and *sample size*. For example, suppose a query is executed with the following parameters: *jump size* of 10, *tuples per peer* 100, and a *sample size* of 400. After selection of the first peer, at least 30 hops are required by the random walk before completion. For the first peer selected p_1, the *period* p_{1_period} is equal to 30 hops. This determines that for the next 30 hops, further processing can be utilized to improve the sample quality for the selected peer p_1. Thus, for each consecutive peer selected, the *period* p_{i_period} is bounded as the (jump size $\times$ the number of remaining peers to be selected).

As shown in Fig. 1, the earlier that a peer is selected for sampling, the larger is the *period* available for gossiping. As additional hops are taken to reach the next peer for sampling, already selected peers can continue to gossip (this is represented by the rings around peers for each *period*). By combining our knowledge of *Gossip* and the p_{i_period} for selected peers, we can maximize the quality of the sample obtained from individual peers. Our hybrid sampling algorithm executes as follows:

1. Given a random start-location peer p_0, the local group is $\{p_0\}$.
2. Initialize a group for each selected peer p_i: $group_i \in \{p_i\}$.
3. For each peer in $group_i$, randomly select r_a adjacent peers.
4. Extend the local group to include adjacent peers if and only if (path from $p_i \leq r_r$) $group_i \in group_i \cup \{\text{for each peer in } group_i \text{ add } p_{i1} \ldots r_a\}$.
5. Perform Gossip on current $group_i$.
6. Continue steps 2-5 for each peer in $group_i$ until p_{i_period} has been reached.
7. All peers selected by the random sampling phase, excluding peers selected by the local groups, send their current mixed values back to the query node.
8. Compute remaining algorithm normally.

Peers near the beginning of the random walk have a longer *period* to *gossip*, whereas peers closer to the end of the walk contain an incrementally smaller *period* for *gossiping*. This creates an uneven level of mixing among the local groups of peers, but since all peers obey *mass conservation* as previously defined, the number of rounds performed by each group does not affect the overall results between the different gossiping groups.

5.2 Benefits and Drawbacks

Our hybrid approach allows a single random walk to increase the quality of samples from selected peers by allowing limited diffusing of values with surrounding neighbors. This technique builds upon our original algorithm by increasing the quality of individual samples, providing a better representation of the network (spurious/unrepresentative data is diffused). In addition, our technique requires no additional messages to be sent to the query

node. Our goal is not to ensure diffusion at a specific rate but instead to allow the *Gossip* protocol to execute online in conjunction with the random sampling process. This gives us a performance speedup, with no additional time requirements for processing beyond the time required by the original random-walk phase, allowing more individual peers to be evaluated without increasing the total number of messages sent back to the query node.

Inversely, our hybrid approach has a high in-network communication cost as the gossip radius increases: As the number of peers participating in the gossiping process increases, the number of messages sent over the network increases rapidly. The hybrid approach incurs no additional messages to be sent to the query node, but there are an increased number of messages sent in-network between gossiping peers. Combined with two gossip-limiting parameters r_a and r_r, we provide a tunable technique by providing a powerful trade-off between the number of messages sent over the network and accuracy.

6 Experimental Evaluation

In this section, we have provided experimental justification for our methods. We have implemented our algorithms on simulated and real-world topologies by using various degrees of data clustering and topology structures.

6.1 Implementation

Our algorithms and P2P topologies are implemented in Java 5.0 with the graph generation tool Jung [20] version 1.6. Our implementation includes both sampled and real-world Gnutella topology samples. All of our experiments were run on AMD dual-core Opteron 2.0-GHz processors with 2 Gbytes of RAM.

6.2 Generation of P2P Networks and Databases

6.2.1 Synthetic Topology

The power laws [14] offer insight to the structure of Internet topologies, and [2] confirms that the power laws extend to P2P networks. Our synthetic topology is created through the process of connecting subgraphs by using the graph generation tool Jung [20]. It consists of 10,000 peers and 100,000 edges. The parameters during graph creation are

- **Subgraphs** (s). s subgraphs are created, which follow the power-laws topology [14].
- **Edges between subgraphs** (e). The size of e determines the cut size between subgraphs. As the cut size decreases, the number of edges between subgraphs decreases.

6.2.2 Real-World Topology

We also experimented with 2001 Gnutella topology data containing 22,556 peers and 52,321 edges, acquired from the group of M. Ripeanu at the University of Chicago.

6.3 P2P Databases

Both types of networks were populated with data generated by a synthetic data generator. We use single-attribute tuples. The attribute values have a range between 1 and 100. The values follow the Zipf distribution. The parameters that define the main characteristics of our synthetic data sets are listed as follows:

- **Cluster Level (CL)**. If the CL is equal to 0, then the data set is perfectly clustered; that is, it is sorted and then partitioned across the peers. If the CL is set to 1, then the data set is randomly permuted then partitioned across the peers. In-between values correspond to in-between scenarios.
- **Skew (Z)**. The skew determines the slant in frequency distribution of distinct values in the data. Low skew values give the data set an even distribution of frequencies per value; conversely, high skew values distort the distribution of frequencies.

We populated the synthetic network with 1,000,000 tuples and the Gnutella network with 2,200,000 tuples. It is well known that P2P databases have strong clustering properties; for example, large networks such as Gnutella contain subgraphs of peers containing similar music genre, movies, software, or documents [25]. Thus, while populating the peers of both networks, we distributed the data in a breadth-first method in order to obtain reasonable clustering of synthetic data within the topologies. That is, when loading a peer, the adjacent peers are also loaded with similarly clustered data.

6.4 Input Parameters

We evaluate the accuracy, the use of network resources, the size of sample acquired, and the total number of tuples sampled from the network. We define each of the user defined inputs as follows:

1. **Required Accuracy** (Δ_{req}). This parameter defines the maximum allowed error for the estimated answer.
2. **Tuples Sampled per Peer** (t). This parameter defines the number of tuples to be sampled from each selected peer.
3. **Jump Size** (j). This parameter defines the number of peers to be passed over before selecting the next peer for sampling.
4. **Initial Sample Size** (r_{orig}). This parameter defines the initial number of tuples to be acquired from the database to execute the first phase. (Thus, $r_{orig}/t = m$, where m is the number of peers visited in the first phase. In our experiments, the local databases are always large enough to ensure that subsampling always takes place.)

Parameter 1 is provided by the user for each query. Parameters 2-4 may be provided by the user or may be set via a preprocessing step.

6.5 Evaluation Metrics: Cost and Accuracy

Our algorithms are evaluated based on the cost of execution and how close they get to the desired accuracy. As discussed earlier, we use latency as a measure of our cost, noting that in our case, it is proportional to the number of peers participating in the random walk. In fact, if the number of tuples to be sampled is the same for all peers (which is true in our experiments), then latency is also proportional to the total number of sample tuples drawn by the overall algorithm. Thus, we use the number of sample tuples used as a surrogate for latency in describing our results.

6.6 Experiments

All of our results were generated from five independent experiments and averaged for each individual parameter configurations. Errors are normalized between 0 and 1.

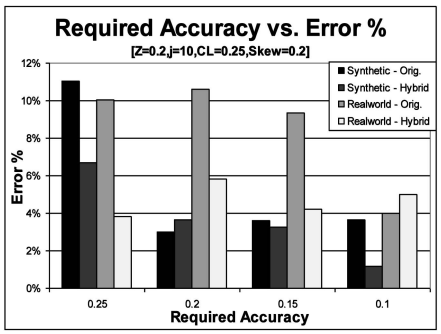

Fig. 2. Effects of required accuracy on the error percentage for the *COUNT* technique.

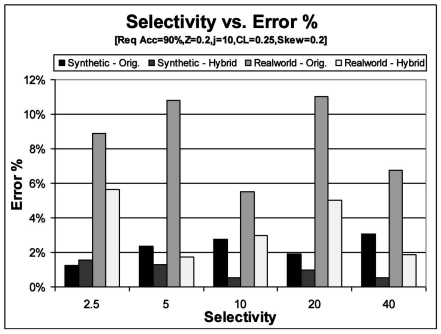

Fig. 3. Effects of selectivity on the error percentage for the *COUNT* technique.

6.6.1 Accuracy

Figs. 2 and 3 shows representative accuracy results for COUNT by using synthetic and real data sets. In this case, we have a query with selectivity of 30 percent, $\mathrm{CL} = 0.2$, and $\mathrm{Z} = 0.2$. In Fig. 2, we vary the required accuracy. The figure shows that the algorithm's result is always within the required accuracy. In Fig. 3, we set required accuracy to 0.1 and show the resulting accuracy for each query with different selectivities.

6.6.2 Sample Size

Figs. 4 and 5 show that the required sample size increases with $1/\Delta_{req}^2$. They also show that the required sample size does not vary much when the initial sample is ranged from 1,000 to 3,000. The selectivity of the query in this experiment was 30 percent, and the algorithm gave an answer within the required accuracy in all cases. We note that the result of our algorithm specifies the number of peers to be sampled. In the experiments, we convert it to the number of samples by taking 25 samples per peer. Fig. 6 shows that the improvement by getting more tuples per peer is small. To minimize the cost of sampling in each peer, we take 25 samples in each peer.

6.6.3 Comparison with Naive Techniques

Fig. 7 compares our approach with depth-first search (DFS), where we collect our sample by using a random walk, with $\mathrm{j} = 0$, and BFS, where we collect our sample from the peers in the neighborhood of the querying peer. Note that our method always meets the required accuracy. Our technique clearly outperforms both techniques.

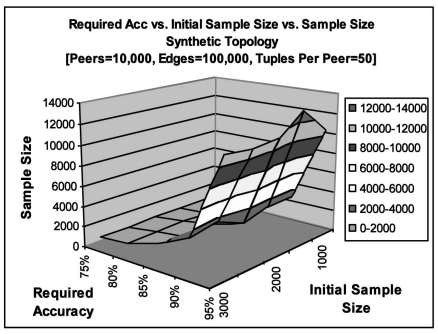

Fig. 4. Effects of the sample size collected for given required accuracies and initial sample sizes for the *COUNT* technique.

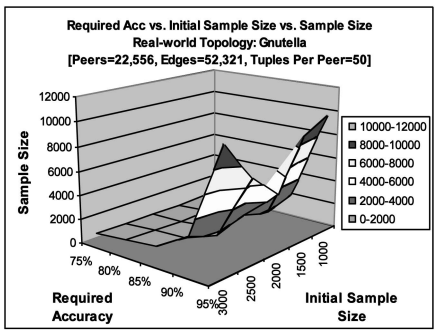

Fig. 5. Effects of the sample size collected for given required accuracies and initial sample sizes for the *COUNT* technique.

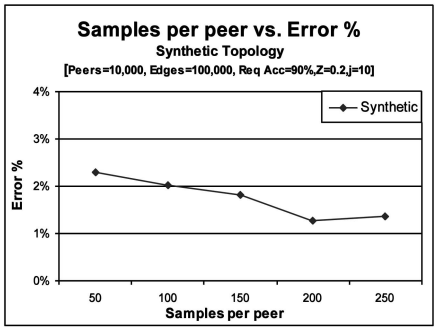

Fig. 6. The number of peers does not make a vast difference in accuracy.

6.6.4 Effects of Data Clustering and Skew

Figs. 8 and 9 show the effects of different degrees of data clustering and Figs. 10 and 11 show different degrees of skew. Figs. 7, 8, 9, 10, 11, and 12 simulate a P2P database with two subgraphs, each containing similar data within individual subgraphs but different from others. The results show that with clustering closer to 0 (data are more clustered), we need to collect more samples, whereas with clustering close to 1 (data are less clustered), we need less samples, since each peer contains a better sample of the entire data set. Regarding skew, the results show that when skew increases, we need fewer samples. The reason is that some values become much more frequent in the data set and, therefore, easier to estimate their count.

6.6.5 Graph Size versus Jump Size

Fig. 12 illustrates the relationship between jump size and cut size in a P2P database. As the number of edges connecting subgraphs or the jump size increases, the

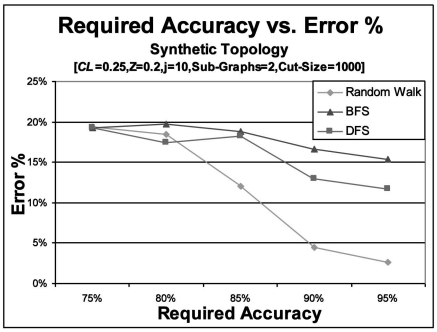

Fig. 7. Random walks perform better than BFS and DFS.

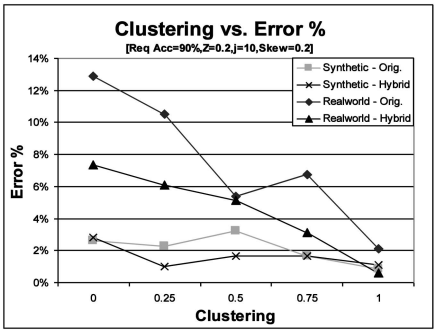

Fig. 8. Effects of clustering on the error percentage for the *COUNT* technique.

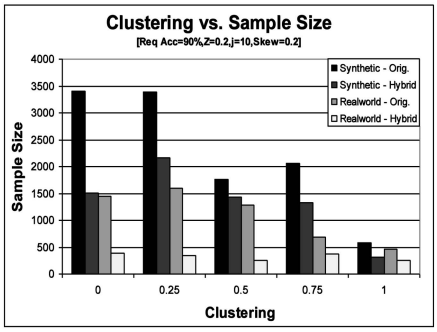

Fig. 9. Effects of clustering on the sample size for the *COUNT* technique.

accuracy of the sample increases. The relationship between number of edges connecting subgraphs and the jump size are inversely proportional in determining the quality of the sample acquired.

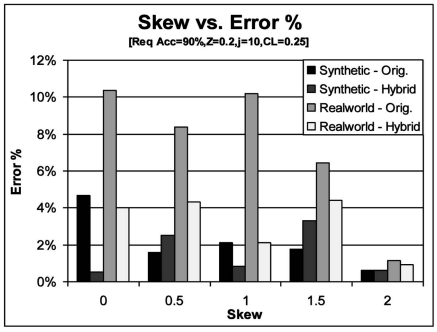

Fig. 10. Effects of skew on the error percentage for the *COUNT* technique.

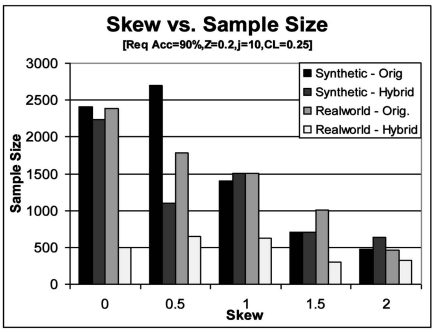

Fig. 11. Effects of skew on the sample size for the *COUNT* technique.

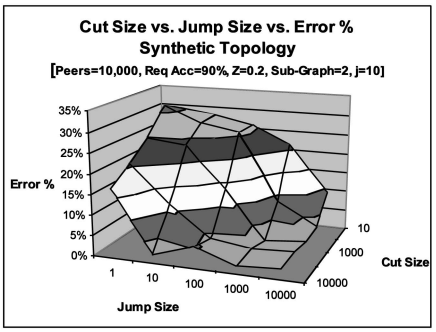

Fig. 12. Effects of cut size with jump size on error percentage for *SUM* technique.

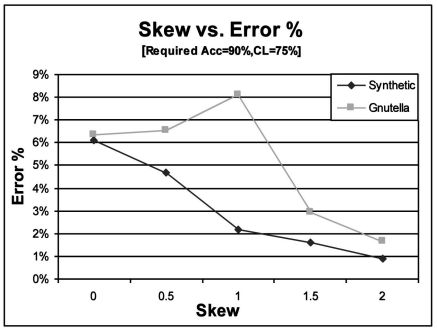

Fig. 13. Effects of clustering on the error percentage for the *SUM* technique.

6.6.6 Evaluating the SUM Query

Figs. 13 and 14 show that our technique provides similar accuracy results for SUM. Here, we estimate the SUM of all tuples in the database (that is, $\text{selectivity} = 1$).

6.6.7 Clustering versus the Number of Tuples Sampled per Peer

Figs. 15 and 16 illustrate the relationship between the clustering of data in the network and the number of tuples selected from each peer. As the clustering of the data increases, the estimated accuracy of the algorithm decreases. In addition, as the number of tuples sampled from individual peers increases, the accuracy decreases steadily. This shows that the ability to acquire larger samples from fewer peers is highly dependent upon the clustering of the data. As shown in Figs. 17 and 18, there is a direct relationship between clustering and the number of tuples sampled from each peer: Highly clustered networks can be better estimated using smaller sample sizes per peer; inversely, networks with little or no clustering can be

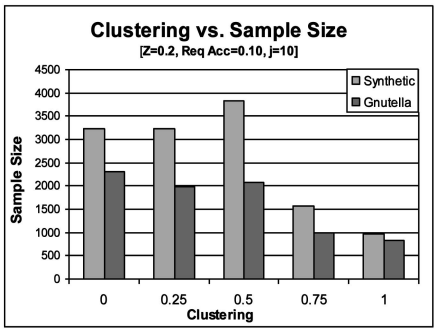

Fig. 14. Effects of clustering on the sample size for the *SUM* technique.

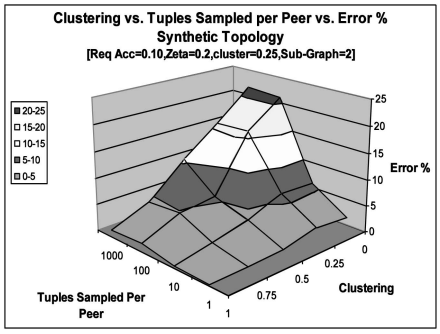

Fig. 15. Effects of clustering with the number of tuples sampled per peer on the error percentage by using the *AVERAGE* technique for synthetic topology.

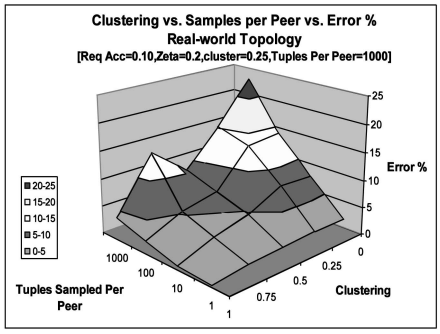

Fig. 16. Effects of clustering with the number of tuples sampled per peer on the error percentage by using the *AVERAGE* technique for real-world topologies.

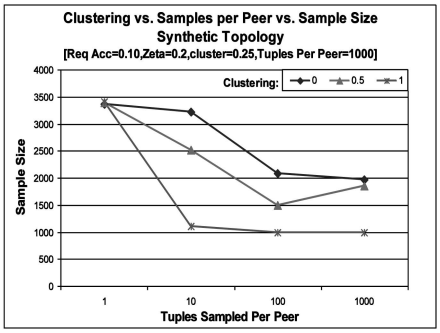

Fig. 17. Effects of clustering with the number of tuples sampled per peer on the sample size by using the *AVERAGE* technique for synthetic topologies.

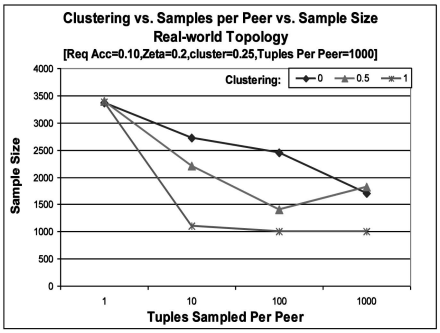

Fig. 18. Effects of clustering with the number of tuples sampled per peer on the sample size by using the *AVERAGE* technique for real-world topologies.

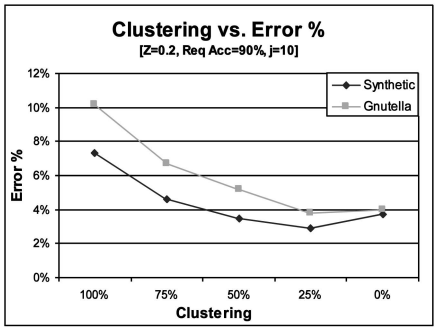

Fig. 19. Effects of clustering on the error percentage for the *MEDIAN* technique.

estimated using larger samples sizes per peer, reducing the total number of messages sent over the network.

6.7 Estimating the MEDIAN

Figs. 19 and 20 illustrate that our technique can be extended to accurately estimate the MEDIAN. Similarly to SUM and COUNT aggregates, our technique for computing the MEDIAN performs well by using various levels of clustering. In these experiments, we use both the Gnutella and the synthetic graphs, vary the clustering factor, and set $\Delta_{req} = 0.1$. The error that we show in the graph is the difference between the true rank of the median that the algorithm returns and $N/2$.

6.8 Hybrid Random Sampling Technique

Our sampling technique, as shown in Figs. 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, and 20, can be extended to include in-network sampling, as proposed in Section 5. Figs. 2, 3, 8, 9, 10, and 11 show that the hybrid approach achieves good results for (synthetic and real-world topologies) in a variety of settings including skew, data clustering, and selectivity. As shown in Figs. 8 and 9, the hybrid solution is effective for data sets with various levels of clustering. In all cases, the hybrid approach outperforms the original algorithm, achieving higher accuracy with a lower sample size. Similar results are shown for the skew in Figs. 10 and 11.

In Figs. 21 and 22, we have included the original algorithm by setting the radius r_r to 0, which represents the original algorithm without gossiping. As shown in Fig. 21, as the number of edges that a peer randomly selects for gossiping r_r and the maximum number of hops from p_i that gossiping is permitted r_a are increased, the accuracy of the results increases steadily. As shown in Fig. 21, as r_r and r_a are increased, the accuracy increases for the hybrid approach, obtaining lower error percentage by as much as 19 percent as compared to the original algorithm. In addition, Fig. 22 illustrates that as r_r and r_a are increased, the required sample size decreases. In some cases, the

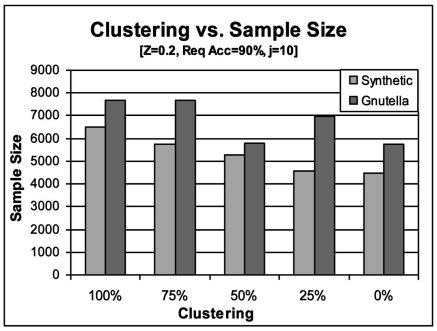

Fig. 20. Effects of clustering on the sample size for the *MEDIAN* technique.

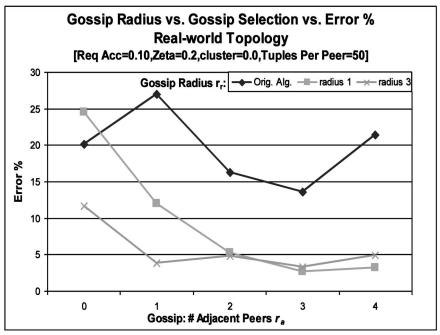

Fig. 21. Effects of Gossip radius with number of adjacent peers on error percentage for the *AVERAGE* technique.

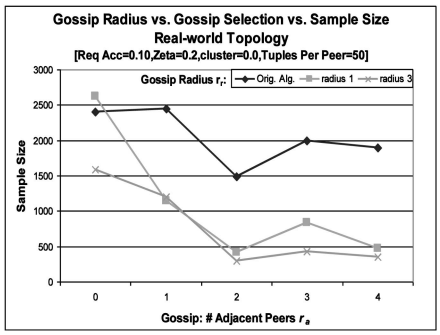

Fig. 22. Effects of Gossip radius with number of adjacent peers on error percentage for the *AVERAGE* technique.

hybrid approach requires more than 1,000 tuples than the original approach. As shown in Figs. 21 and 22, small values for r_r and r_a achieve improved results.

7 Conclusion

In this paper, we present adaptive sampling-based techniques for the approximate answering of ad hoc aggregation queries in P2P databases. Our approach requires a minimal number of messages sent over the network and provides tunable parameters to maximize performance for various network topologies.

Our approach provides a powerful technique for approximating aggregates of various topologies and data clustering but comes with limitations based upon a given topologies structure and connectivity. For topologies with very distinct clusters of peers (small cut size), it becomes increasingly difficult to accurately obtain random samples due to the inability of random-walk process to quickly reach all clusters. This can be resolved by increasing the *jump size*, allowing a larger number of peers to be considered and increasing the allowed mixing by our hybrid approach. By varying a few parameters, our algorithm successfully computes aggregates within a given required accuracy. We present extensive experimental evaluations to demonstrate the feasibility of our solutions for both synthetic and real-world topologies.

Acknowledgments

The authors thank Matei Ripeanu of the University of Chicago for providing us with the Gnutella topology samples. Real-world topology samples are available at http://people.cs.uchicago.edu/~matei/GnutellaGraphs. The work of Gautam Das was supported by unrestricted gifts from Microsoft Research and start-up funds from the University of Texas, Arlington. The work of Dimitrios Gunopulos was supported by IIS-0330481. The work of Vana Kalogeraki was supported by CNS-0627191 and IIS-0330481.

References

[1] S. Acharya, P.B. Gibbons, and V. Poosala, "Aqua: A Fast Decision Support System Using Approximate Query Answers," *Proc. 25th Int'l Conf. Very Large Data Bases (VLDB '99),* 1999.

[2] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman, "Search in Power-Law Networks," *Physical Rev. E,* 2001.

[3] B. Babcock, S. Chaudhuri, and G. Das, "Dynamic Sample Selection for Approximate Query Processing," *Proc. 22nd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '03),* pp. 539-550, 2003.

[4] A.R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," *Proc. ACM Ann. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '04),* 2004.

[5] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Analysis and Optimization of Randomized Gossip Algorithms," *Proc. 43rd IEEE Conf. Decision and Control (CDC '04),* 2004.

[6] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip and Mixing Times of Random Walks on Random Graphs," *Proc. IEEE INFOCOM '05,* 2005.

[7] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya, "Towards Estimation Error Guarantees for Distinct Values," *Proc. 19th ACM Symp. Principles of Database Systems (PODS '00),* 2000.

[8] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. Narasayya, "Overcoming Limitations of Sampling for Aggregation Queries," *Proc. 17th IEEE Int'l Conf. Data Eng. (ICDE '01),* pp. 534-542, 2001.

[9] S. Chaudhuri, R. Motwani, and V. Narasayya, "Random Sampling for Histogram Construction: How Much Is Enough," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '98),* pp. 436-447, 1998.

[10] S. Chaudhuri, G. Das, and V. Narasayya, "A Robust Optimization-Based Approach for Approximate Answering of Aggregate Queries," *Proc. 20th ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '01),* 2001.

[11] S. Chaudhuri, G. Das, and U. Srivastava, "Effective Use of Block-Level Sampling in Statistics Estimation," *Proc. 23rd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '04),* 2004.

[12] Y. Chu, S. Rao, and H. Zhang, "A Case for End System Multicast," *Proc. ACM Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '00),* 2000.

[13] M.M. Espil and A.A. Vaisman, "Aggregate Queries in Peer-to-Peer OLAP," *Proc. Seventh ACM Int'l Workshop Data Warehousing and On-Line Analytical Processing (DOLAP '04),* 2004.

[14] C. Faloutsos, P. Faloutsos, and M. Faloutsos, "On Power-Law Relationships of the Internet Topology," *Proc. ACM Ann. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '99),* 1999.

[15] Freenet Homepage, http://freenet.sourceforge.net, 2006.

[16] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks in Peer-to-Peer Networks," *Proc. IEEE INFOCOM '04,* 2004.

[17] Gnutella Homepage, http://rfc-gnutella.sourceforge.net, 2006.

[18] P. Haas and C. Kőnig, "A Bilevel Bernoulli Scheme for Database Sampling," *Proc. 23rd ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '04),* 2004.

[19] R. Heubsch, J. Hellerstein, N. Lanhan, B.T. Loo, S. Shenker, and I. Stoica, "Querying the Internet with PIER," *Proc. 29th Int'l Conf. Very Large Data Bases (VLDB '03),* 2003.
[20] JUNG Web Site, http://jung.sourceforge.net, 2006.
[21] P. Kalnis, W.S. Ng, B.C. Ooi, D. Papadias, and K.-L. Tan, "An Adaptive Peer-to-Peer Network for Distributed Caching of OLAP Results," *Proc. 21st ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '02),* 2002.
[22] Kazaa Homepage, http://www.kazaa.com, 2006.
[23] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-Based Computation of Aggregate Information," *Proc. 44th Ann. IEEE Symp. Foundations of Computer Science (FOCS '03),* 2003.
[24] V. King and J. Saia, "Choosing a Random Peer," *Proc. 23rd Ann. ACM Symp. Principles of Distributed Computing (PODC '04),* 2004.
[25] F. Le Fessant, S. Handurukande, A.-M. Kermarrec, and L. Massoulié, "Clustering in Peer-to-Peer File Sharing Workloads," *Proc. Third Int'l Workshop Peer-to-Peer Systems (IPTPS '04),* 2004.
[26] W. Lee, S.J. Stolfo, and K.W. Mok, "A Data Mining Framework for Building Intrusion Detection Models," *Proc. IEEE Symp. Security and Privacy,* p. 0120, 1999.
[27] X. Li, Y.J. Kim, R. Govindan, and W. Hong, "Multi-Dimensional Range Queries in Sensor Networks," *Proc. First ACM Int'l Conf. Embedded Networked Sensor Systems (SENSYS '03),* 2003.
[28] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," HP Technical Report HPL-2002-57, 2002.
[29] B. Mukherjee, L. Heberlein, and K. Levitt, "Network Intrusion Detection," *IEEE Network,* vol. 8, no. 3, pp. 26-41, May/June 1994.
[30] Napster Homepage, http://www.napster.com, 2006.
[31] G. Pandurangan, P. Raghavan, and E. Upfal, "Building Low-Diameter P2P Networks," *Proc. 42nd Ann. IEEE Symp. Foundations of Computer Science (FOCS '01),* 2001.
[32] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. ACM Ann. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '01),* 2001.
[33] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms (Middleware '01),* 2001.
[34] O.D. Sahin, A. Gupta, D. Aggrawal, and A. El Abbadi, "A Peer-to-Peer Framework for Caching Range Queries," *Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04),* 2004.
[35] S. Sen, O. Spatscheck, and D. Wang, "Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures," *Proc. 13th Int'l World Wide Web Conf.,* 2004.
[36] J.L. Simon, *Resampling: The New Statistics,* second ed., Oct. 1997.
[37] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM Ann. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '01),* 2001.
[38] D. Zeinalipour-Yazti, V. Kalogeraki, and D. Gunopulos, "Exploiting Locality for Scalable Information Retrieval in Peer-to-Peer Networks," *Information System,* vol. 30, no. 4, pp. 277-298, 2005.
[39] J. Zhao, R. Govindan, and D. Estrin, "Computing Aggregates for Monitoring Wireless Sensor Networks," *Proc. First IEEE Int'l Workshop Sensor Network Protocols and Applications (SNPA '03),* 2003.

Benjamin Arai received the BSc degree in computer science from the University of California, Riverside, in 2004. He is a PhD candidate in the Department of Computer Science and Engineering, University of California, Riverside. His research interests include data mining and knowledge discovery, databases, peer-to-peer networks, top-k retrieval, and sensor networks. He is a student member of the IEEE.

Gautam Das received the PhD degree in computer science from the University of Wisconsin, Madison, in 1990. He has been an associate professor in the Department of Computer Science and Engineering Department, University of Texas, Arlington, since 2004. He has held positions at the University of Memphis, Compaq Corp., and, most recently, Microsoft Research. In addition, he has held visiting positions at the Max-Planck-Institut for Informatics, Germany, the University of Helsinki, and the Indian Institute of Science, Bangalore, India. He has served on numerous program committees and was the program committee cochair of the First International Workshop on Ranking in Databases (DBRank '07) and the Ninth ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD '04). He is currently a guest editor of the *ACM Transactions on Knowledge Discovery from Data*. His research interests include data mining and knowledge discovery, databases, algorithms, and computational geometry. His research has been supported by the US National Science Foundation (NSF), the US Office of Naval Research (ONR), Microsoft, Cadence Design Systems, and Apollo Data Technologies.

Dimitrios Gunopulos received the PhD degree from Princeton University in 1995, and since then, he has been a postdoctoral fellow at the Max-Planck-Institut for Informatics, Germany, and a research associate at the IBM Almaden Research Center. He has been a professor in the Department of Computer Science and Engineering, University of California, Riverside, since December 1998. He has served as a program committee cochair of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD '06) and the 15th International Conference on Scientific and Statistical Database Management (SSDBM '03) and is currently an associate editor of the *IEEE Transactions on Knowledge and Data Engineering* and the *ACM Transactions on Knowledge Discovery from Data*. His research interests include data mining and knowledge discovery, databases, and algorithms. His research has been supported by the US National Science Foundation (NSF), including a Faculty Early Career Development (CAREER) award, the US Department of Defense (DoD), the Institute of Museum and Library Services, the Tobacco Related Disease Research Program, and AT&T. He is a member of the IEEE.

Vana Kalogeraki received the PhD degree from the University of California at Santa Barbara in 2000. She is an assistant professor in the Department of Computer Science and Engineering, University of California, Riverside. She has worked as a research scientist in Hewlett-Packard Laboratories. She has served as the program cochair of the VLDB International Workshop on Databases, Informations Systems and Peer-to-Peer Computing (DBISP2P '03), the 13th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS '05), the IEEE International Conference on Pervasive Services (ICPS '05), and the 10th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC '07) and the general cochair of WPDRTS '06. She is currently an associate editor of the *Ad Hoc Networks Journal*, the *Computer Standards and Interfaces Journal*, and the *Peer-to-Peer Networking and Applications Journal*. Her research interests include distributed and real-time systems, peer-to-peer systems, and sensor networks. Her research is supported by the US National Science Foundation. She has published many technical papers and given many tutorials. She is a member of the IEEE.