Distinct Value Estimation on Peer-to-Peer Networks

Zubin Joseph
UT Arlington
zubin.joseph@uta.edu

Gautam Das
UT Arlington
gdas@cse.uta.edu

Leonidas Fegaras
UT Arlington
fegaras@cse.uta.edu

ABSTRACT
Peer-to-Peer networks have become very popular on the Internet, with millions of peers all over the world sharing large volumes of data. In the assistive healthcare sector, it is likely that P2P networks will develop that interconnect and allow the controlled sharing of patient databases of various hospitals, clinics, and research laboratories. However, the sheer scale of these networks has made it difficult to gather statistics that could be used for building new features. In this paper, we present a technique to obtain estimations of the number of distinct values matching a query on the network. We evaluate the technique experimentally and provide a set of results that demonstrate its effectiveness, as well as its flexibility in supporting a variety of queries and applications.

1. INTRODUCTION
Peer-to-Peer (P2P) networks are a highly popular medium for sharing CPU processing power, storage space, and/or content in the form of text documents and various forms of media. Some applications such as Skype [12] even use such networks for Voice over IP (VoIP) telephony. In the assistive healthcare sector, it is likely that P2P networks will develop that interconnect and allow the controlled sharing of patient databases of various hospitals, clinics, and research laboratories. These networks usually operate over the Internet and consist of thousands, and even millions of peers that can be located anywhere in the world.

Additionally, P2P networks are designed to be scalable, fault-tolerant and dynamic with no central point of failure. In the unstructured peer-to-peer networks that we focus on in this paper, peers do not make any assumptions about the location of other peers, the data, or of the network topology. Each peer maintains connections to a small set of neighbors that are usually accessed over the Internet through their IP addresses.

Each peer on the network contributes resources that are accessible by other peers on the network. This contribution is often in the form of music files, especially in popular P2P networks such as Gnutella [10] and KazaA [13]. All peers are considered ‘equal’, in that they can be either servers or clients, depending on the services or resources that they are providing or accessing. Additionally, peers are free to join and leave the network at any time.

Given the volatility of this architecture and the vast numbers of participating peers, it is often difficult to keep track and gather statistics of the large volumes of the data that is available on the P2P network. In this paper, we focus on a technique that samples a subset of peers in order to estimate the total number of *distinct* tuples that match a query on the network.

The problem of distinct value estimation is well known in the domain of databases. The capability of answering queries from any peer allows trends and data mining to be inferred by making use of density and duplication values – e.g., the spreading patterns of certain diseases can be mined by gathering such statistics across distributed health case databases. It is especially important for query optimization, especially in the construction of histograms [1, 4, 14, 15]. Histograms use the number of distinct values of an attribute (in a table/bucket) to maintain statistics, such as the density, which is the average number of duplicates per distinct value.

1.1 Potential Applications
Currently, distinct value estimation techniques do not exist for P2P networks. Having this type of statistics available would not only allow histogram construction in the future, but would also enable management, administration, monitoring, and report generation capabilities to be built into such systems. For example, on a healthcare P2P network, queries such as the total number of distinct diseases in a geographical region, or the total number of distinct patients suffering from a particular ailment could be estimated for the entire network. This capability even allows trends to be gathered by making use of density and duplication values.

New and exciting new applications also become possible. Consider a peer-to-peer network where each user at a peer submits queries to the network or to a database. Distinct value estimation can be used to assess the query logs of users and discover the number of unique queries about a certain topic or subject. This too, can be used to assess popularity of queries.

1.2 Challenges
The problem of distinct value estimation on unstructured peer to peer networks is a new and, to the best of our knowledge, has not been investigated to date.

Performing such estimations on P2P networks has many challenges. One of the most apparent difficulties is that distinct value estimation in a centralized data repository is known to be a hard problem, as proved in [4]. A variety of estimators [4,1 4, 15] exist, but currently none can provide guaranteed bounds for error for a uniform-random sample of the tuples in the column of a table [1].

Additionally, this problem is made more challenging by the fact that our estimation is targeted toward queries executed on unstructured peer-to-peer networks, where each peer only knows information about its neighbors. More specifically, in this architecture, no peer has knowledge of the topology of the P2P network, the distribution of the data residing on the network , nor of the location or sizes of all other peers on the network.

With constraints such as these, it is a challenge for a user at one of these peers to obtain a uniform random sample to apply a distinct value estimator. There is also an associated network cost in accessing and retrieving samples from other nodes across the network. This must be minimized to avoid long lag times before users see the results of a distinct value approximation query.

The calculations for gathering such statistics may require parameters such as the total size of the data available on the network or the total number of nodes. Parameters such as these may be largely unavailable at a peer, and may also need to be approximated.

Distinct value estimation queries are typically made for one or more attributes in a table. In a typical P2P network, such as Gnutella, these attribute values have been shown to follow a Zipfian distribution [30, 21], with duplicates residing on different peers scattered throughout the network. This potential clustering of similar data at neighboring peers adds another interesting dimension to the problem. Furthermore, the distributions and clustering levels vary depending on the attribute under consideration. The range of queries that are possible makes distinct value estimations even more compelling.

1.3 Overview of Our Algorithm

In this paper, we offer a technique of performing distinct value estimations that is a novel combination of strategies from different domains. Our algorithm operates as follows.

It executes a Metropolis-Hastings[3] random walk sampler on the P2P network in order to randomly select peers from the entire set of peers on the system. It then obtains a uniform-random data sample from each selected peer. Then, using a block-level sampling technique (called COLLAPSE [1]), it removes duplicates from within a node-sample. After combining the collapsed samples from multiple peers at the sink (the peer that originated the query), it applies an existing distinct value estimator by treating the entire sample as a single uniform-random one.

1.4 Our Contributions

In this paper, we show that by leveraging these different techniques, our algorithm for distinct value estimation is very effective because:

- It does not require that peers exchange calculations or knowledge of constants that govern algorithm behavior.
- It reduces the preprocessing time, and the number of assumptions and approximations of P2P network characteristics, such as the total size, number of nodes, etc.
- It minimizes the information that a node needs to maintain about its neighbors.
- It is largely independent of the clustering and/or distribution characteristics of the data in the P2P network.
- It allows the flexibility of changing distinct value estimators.
- Its performance, in some cases, can approach that of a uniform random sample of the entire dataset of the network.

The rest of this paper is organized as follows. In Section 2 we discuss related work. In Section 3 we discuss the key ideas that are leveraged in our eventual algorithm for computing distinct values over P2P databases. The actual algorithm is described in Section 4. In Section 5 we show how our algorithm can be applied to a P2P network to answer simple queries. In Section 6 we describe a comprehensive set of experiments that demonstrate the effectiveness of our approach. We conclude in Section 7.

2. RELATED WORK

A variety of search and node traversal techniques exist for peer to peer networks, both structured and unstructured. Many these are surveyed and described in great detail in [16, 17].

A vast amount of studies have been done on random walks including [5, 24]. The Metropolis-Hastings algorithm is also discussed in [3], and is used to execute a random walk over documents indexed by a search engine. Alternatives are also suggested in [25], where the authors suggest the Random Weight Distribution method, which requires underlying support of the P2P network. The authors in [11] suggest an interesting modification to Metropolis-Hastings sampler that make it suitable for dynamic graphs. In this paper we only address static graphs. We leave handling highly dynamic cases as future work.

Using sampling for estimating query results is a well known problem that has received a large amount of attention by Haas et al [32], Lipton et al [31] and Hou et al [22, 23]. Sampling has also been used for approximate query processing in centralized databases [1, 14, 35, 36, 24, 4].

Several authors have looked into approximation-type queries for P2P networks, including using random walks over the web in [18], and aggregations over unstructured P2P networks as in [2]. Alternative gossip-style techniques of computing aggregates have been suggested by [20], but require participation of every node in the system. Techniques utilizing structured peer-to-peer networks have addressed the problem of sampling random peers [19], as well as approximations [26], and there is a large body of work on these types of networks. In this paper we target unstructured P2P networks, which require a different approach.

In statistical literature, cluster sampling is a concept considered similar to block-level sampling [1]. In this paper we consider this technique of sampling, originally proposed in [22, 23]. An analysis of block-level sampling as applied to databases is discussed in [1], where block-level estimates are used for histogram construction and distinct value estimation. Chaudhuri et al consider this in [14], and also suggest an optimal error distinct value estimator.

Other Distinct Value Estimators have been proposed as well, such as the Adaptive Estimator [4], the Goodman Estimator [27] and other estimators in [15, 28, 29].

3. FOUNDATIONS OF OUR APPROACH

In this section, we provide the foundations of our approach to this novel problem. Our actual algorithm is discussed in the next section.

We model the P2P network as a graph G with peers as nodes and edges connecting nodes to their neighbors. We therefore refer to nodes and peers interchangeably throughout this paper. We assume that each peer P_i has a local database D_i, and refer to the total data D on the network including duplicates, as a multi-set as it consists of the sets of data residing at each peer in the network. We consider SQL-like queries of the form "SELECT COUNT (DISTINCT *) FROM D WHERE <selection condition>". As indicated in the introduction, such queries can be extremely important for statistics estimations in many emerging applications. Our objective is to obtain the best estimates of distinct values possible, preferably within a given bound on the cost (or latency) of executing the query. In general, the cost of query execution is dependent on the cost of traversing the network to sample peers, as well as the cost of sampling the local databases and sending the data back to the originating peer for the result estimation.

3.1 Distinct Value Estimators

We discuss two distinct value estimators, the Guaranteed-Error Estimator (GEE) [14] and the Adaptive Estimator [4]. Both of these estimators require a uniform-random sample and the counts (f_i values) of the elements that occur i times in the sample of r elements. In order to estimate the total number of distinct elements ($\hat{D}$), both of these estimators only scale the number of single occurrences of elements in a uniform-random sample. Multiple occurrences of an element are not scaled.

3.1.1 Guaranteed Error Estimator

The Guaranteed Error Estimator (GEE) [4, 14] is an estimator with optimal error and a bias of at most $O(\sqrt{1/q})$ [1, 4, 14].

$$\hat{D} = \sqrt{\frac{n}{r}} f_1 + \sum_{j=2}^{r} f_j$$

where f_i is the number of distinct elements in the sample that occur i times. It requires that the value of n is known, where n is the size of the set that contains all the values of an attribute in a table, including all duplicates. In our case, n refers to the total number of data tuples in the entire network.

3.1.2 Adaptive Estimator

The Adaptive Estimator (AE) [4] is a heuristic estimator for the number of distinct values. It takes on the form:

$$\hat{D} = Kf_i + d$$

where d is the number of distinct values in the sample, given by $d = \sum_{i=1}^{r} f_i$ and K is an appropriate scaling factor that is computed from the sample. An important feature of AE is that unlike GEE, it does not require a value for n (the total size of the multi-set).

3.1.3 Error Metrics

In this paper, we define the estimation error according to the ratio-error metric given in [4]:

$$Error = \max\left(D / \hat{D}, \hat{D} / D\right)$$

3.2 COLLAPSE

The above estimators assume the availability of a uniform-random sample. However, as we have indicated earlier, obtaining a pure uniform random sample of a dataset distributed across an unstructured P2P network is extremely difficult. Consequently our approach is to obtain a random sample of the set of peers, and from the sampled peers, to obtain random samples of the local databases. To enable adapting the above estimators to work for such samples, we consider a technique proposed in [1] called COLLAPSE, which has been used in the context of distinct value estimation in centralized databases where the sampling is restricted to a uniform random sample of the disk blocks that constitute the underlying table.

COLLAPSE operates by sampling blocks from the database. Duplicates of a value within a block-level sample are *collapsed* into just a single occurrence within the sample. The collapsed sample is then treated like a uniform random sample, and existing estimators, such as the GEE and AE, can be applied to the collective sample from randomly chosen blocks. Such an approach is particularly meaningful in our case as we can model the local databases at peers in a P2P network as the equivalent of disk blocks in a centralized repository.

3.3 Network Traversal

The focus of our paper lies in targeting our estimations to unstructured peer-to-peer networks. Our objective is to obtain the best estimates of distinct values possible with minimal traversal of the network. In this section, we review a few approaches that exist for traversing the network.

3.3.1 Flooding

A simple method of sampling the set of peers is to use flooding. The request initiator sends out messages to all its neighbors, which in turn pass the message on to their neighbors till the message spreads throughout the network to every node. Thus, the initiator will eventually visit every node in the system. However, flooding is highly unsuitable for larger networks and it places great load on participants and on the network [5] due to repeated messages and cycles.

3.3.2 Random Walks

A popular network traversal technique is to use a random walk, where each node picks out a random neighbor to pass its messages instead of passing the messages on to all its neighbors. This technique reduces the network load by an order of magnitude [5] as fewer messages are exchanged between nodes. Using the random walk technique, a chain of nodes is thus built as messages pass from one neighbor to the next. However, this process has a higher latency in returning results than flooding.

Random walks are not an exhaustive technique, and operate on the assumption that tuples that match the query are common on the network.

3.3.2.1 Random Walks and Clusters

In real-world peer-to-peer networks it is inevitable that nodes form clusters. These clusters may have similar data residing on the network. Alternatively, these clusters may form because the nodes may all be in the same geographical area. The formation of clusters causes a problem because if there is a small cut between clusters, then a random walk has a higher chance of accessing more nodes from within a cluster, rather than from across the entire network.

The authors in [2] reduce the effect of this clustering by setting a jump size *j* for the walk, as shown in Figure 1. Only nodes that occur after *j* hops are taken into the sample. With more realistic jump sizes in real P2P networks, this has the effect of making the walk lose memory of recent neighbors since it reduces the correlation between successive sampled peers in the network.

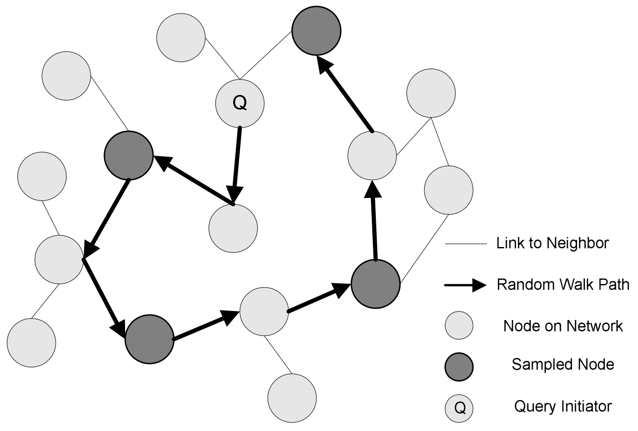

Figure 1: Random Walker with jump size = 2

3.3.2.2 Random Walks and Node Degree

Different nodes in the network may have a different number of neighbors. In Gnutella, some peers may collect many neighbors (called ultrapeers) [16, 17]. Others may just maintain one link to a larger, well-connected node. Because of these differences in node degree, the random walk has a higher probability of traversing a node with large more than one with a smaller degree. This results in a non-uniform sample with a higher chance of duplicates of such nodes.

The problem of varying node degree is addressed in [3] by making use of the random walk traversal technique. This technique makes use of the *Metropolis-Hastings algorithm* [7, 8]. It operates by changing the way a node chooses a neighbor to move to during the random walk process. The process is described as follows.

Consider a node *X* with a total number of neighbors (edges) that is given by *deg*(*X*). Assume that one of these neighbors is node *Y*, which has its own *deg*(*Y*) associated with it. If *X* randomly chooses *Y* as the next node in the random walk then we define the probability that *X* will move to *Y* as:

$$r(X,Y) = \min\left\{1, \frac{\deg(X)}{\deg(Y)}\right\}$$

r(*X,Y*) is called the acceptance probability. Thus, once the node picks out its prospective next peer, it tosses a coin that comes up heads with a probability of *r*(*X, Y*). If it comes up heads it hops to the next node, otherwise it stays at the same peer and performs the coin toss again [3]. Thus, since *r*(*X, Y*) = 1 if deg(*X*) > deg(*Y*), the algorithm will always move towards an *X* and *Y* that satisfy this condition.

The Metropolis-Hastings algorithm gives a higher preference to moving to nodes that have smaller degree (connectivity) than the current node. Thus, this should balance out the fact that lower connectivity nodes normally have a lower probability of being accessed during a normal random walk.

4. OUR ALGORITHM

In this section, we discuss our method of estimating the number of distinct values on an unstructured P2P network. This method uses distinct value estimators on a sample that is built up from executing a random walk on the network and sampling the local databases of the visited peers.

We first discuss how we can apply COLLAPSE to the domain of P2P networks.

4.1 Applying COLLAPSE to P2P Networks

We assume that a set of peers have already been sampled from the network (the specifics of how this is done is discussed in the next sub-section). Here we discuss how the local databases of these nodes are sampled and COLLAPSE is applied to these records.

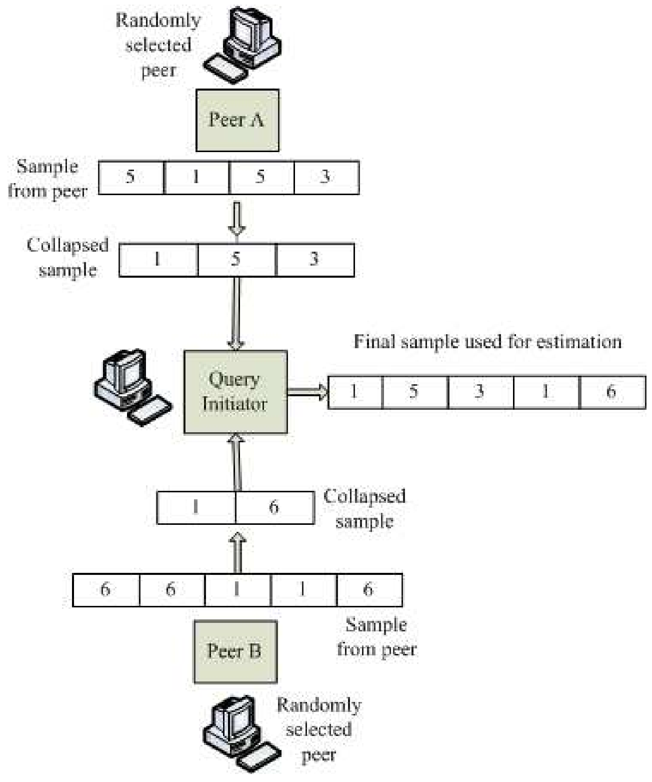

Figure 2: COLLAPSE on P2P Networks

Figure 2 describes how the data sampled from local databases of visited peers are processed through the COLLAPSE algorithm. For each peer, the duplicates are removed from all records that pass the query selection condition, and the resulting dataset is sent over the network to the peer that initiated the query ("sink"). A distinct value estimator is then applied to the accumulated sample records at the sink.

While COLLAPSE was originally applied to centralized databases, it offers a variety of benefits that can also be applicable to peer to peer networks.

Firstly, it is a simple algorithm that requires few additional input parameters. It makes no assumption on the size of each block and it can be applied to different-sized blocks. Since various node sizes are possible in P2P networks, COLLAPSE could be applied to the samples without any fundamental change. Since we can choose how much to sample from each node, this would translate to being able to access fewer nodes in total to build up a sample, by sampling more from each node. This gives power and flexibility to allow the algorithm to be applied in different cases. The sample sizes can also change depending on network conditions, since there is a cost associated with accessing nodes and for retrieving samples from nodes.

Peers on a P2P network are more or less autonomous. They can enter, leave or add and remove shared resources at any time. Because any peer can potentially be an adversary, a proven and robust algorithm that can handle this susceptibility is crucial. COLLAPSE addresses this concern as it is designed to handle such adversarial models [1].

Because COLLAPSE is designed to handle duplication of data across blocks, it factors out the duplications within a block. This preprocessing is important because in P2P networks, it is normal for a node to have multiple records of the same distinct value (if the query is to find distinct values). COLLAPSE eliminates repeatedly counting a single peer that may have a lot of such occurrences of the distinct value.

4.1.1 Node Sampling: Implementation Issues

Depending on the estimator used, the only variable that is required is the total number of sampled records. This is tracked as COLLAPSE executes. Thus, the overall sampling ratio is also taken to be that of the original block sample before running COLLAPSE [1].

If GEE is used for estimation, the value of n must be either estimated from preprocessing or be specified for a particular network/query. However, it should also be noted that AE does not assume that we know n, so we do not have to estimate it in such a preprocessing phase. This would also introduce an additional error in our estimations.

As previous described, the peers are selected during the random walk. They can have different sized databases, samples, and collapsed samples. The collapsed samples from every chosen peer are brought together at the query initiator (sink). The sink also keeps track of the original sizes before collapsing the samples.

4.2 Sub-Sampling Strategies at Nodes

Some nodes in the peer-to-peer network may have too many tuples to be able to send all of their data across the network to the query initiator (sink). Thus, it becomes necessary to sub-sample at each sampled node's local repository. It is important that this node sample be a uniform and random as well.

An important factor to consider is the size of this sample. Since different nodes have different sizes, it is not possible to have a constant size for the sample as this would result in a non-uniform sample of the multi-set. This is intuitive, because if sample size is s and node A is of size s and node B is of size $10s$, then the probability of selecting a tuple at node A is 1, but at node B it is 0.1. This means that not all tuples have the same chance of entering the sample.

The connectivity of nodes also poses a problem to the sampling, since well connected nodes may be picked out for sampling more often. In approximating aggregation queries over P2P networks, the authors in [2] have suggested scaling the contributions from such nodes. This reasoning cannot be applied to distinct value estimation easily as the estimators work on samples that cannot be scaled. An alternative is to reduce the number of samples taken from nodes with high degree. A dynamic sampling ratio such as this poses several challenges and makes it harder to assess the overall sampling ratio and guarantee a bias-free sample.

We opt for a simple sub-sampling technique that uses a constant sampling ratio for nodes sampled during a random walk. This ensures that all tuples residing on sampled nodes are sub-sampled in a fair manner. Thus, larger nodes that may have a significant effect on the rest of the network contribute correspondingly larger samples so that more tuples from such nodes can be represented in the sample.

As previously discussed, an underlying assumption for COLLAPSE to be applied effectively in databases is that blocks are chosen randomly. Thus the selection of nodes for applying COLLAPSE in a P2P network must be as close to random as possible, despite the challenges that arise from the varying connectivity of nodes. To address this, we adopt the use of a random walk sampler that uses the Metropolis-Hastings algorithm [3].

4.3 Metropolis-Hastings Network Traversal

By using the Metropolis-Hastings random walker [3], we aim to obtain a uniform-random sample of nodes from the set of all nodes in the P2P system.

The technique operates upon the assumption of a fixed relationship between the degree and the probability of visiting a peer [11]. This is especially valid since networks such as Gnutella may have ultrapeers that are accessed more frequently during random walks due to their high connectivity.

Instead of scaling contributions based on connectivity as in [2], using the Metropolis-Hastings controls the path of the walker so that a higher preference is given to moving towards nodes with lower connectivity. The outcome of this is a set of random peers that are not necessarily selected because they have high connectivity. This provides our algorithm with the ability to use a constant node sampling ratio across all nodes so that all selected nodes can be processed in a similar manner with the same parameters. This simplifies the operation of our algorithm.

Because this algorithm still operates using a random walk, which is a Markov chain, there is a correlation between the selected nodes since they are along a chain of neighbors. The effect of this is controlled to some degree by using the *burn-in* period parameter.

While the jump size j samples nodes that occur every j hops, the burn-in period b samples the current node after the outcome of b coin-tosses (see [3, 7, 8]. Depending on the result of each coin toss, the walk may or not move to the next peer, and the actual number of hops between sampled nodes ranges between 0 and b.

Setting a higher burn-in period also makes it less likely that the random walker will get stuck sampling nodes with a cluster.

The Metropolis-Hasting method does not require knowledge of any global constants or external settings, other than the burn-in period. All information required at a node to execute the walk, such as the degree of neighbors, can be stored locally or retrieved using a simple query to neighbors. This feature eliminates the need to inform all nodes of system settings.

4.3.1 Random Walk Sampler: Implementation Issues

The optimal value of the burn-in period must be set depending on the application and the topology of the P2P network. It should be large enough to reduce the correlation between sampled nodes. However the larger the number, the higher the latency and wait times due to the increased number of hops between successive sampled nodes.

The random walker must also be designed carefully so that it has a stopping condition; otherwise it would continue executing infinitely.

A simple solution is to execute the walk until the total accumulated sample size reaches a specified ratio of the total data size on the network. As in the GEE estimator (3.1.1), this assumes that *n* is known.

When using estimators such as AE that do not require a value for *n*, it may be preferable to assign a threshold to the random walk. The threshold T operates by assuming we have a cost function combining the time and network costs of accessing a node (via the random walk) and similar costs for retrieving samples across the network from source to sink. Once we model this overall cost of sampling nodes using the walker, the user can define an acceptable threshold at which the walk terminates, i.e., once the cost exceeds the threshold. This approach gives the user the additional flexibility to customize the walker to sample more nodes with a lower sampling ratio or, conversely, fewer nodes with a larger sampling ratio.

4.4 Pseudo-code

We provide simple pseudo-code that defines our algorithm. We first provide the code for the random walk sampler. We do not include details of algorithms of existing work such as COLLAPSE [1], the Metropolis-Hastings sampler [3], nor distinct value estimators.

The random walk is executed over the P2P network, starting with the query initiator node. Note that the sample_size variable is the accumulated value of the original node sample sizes, before applying COLLAPSE. Only the collapsed sample is used in the estimation.

The pseudo-code for the random walk is provided in Figure 3.

```
Inputs:
Rdata          : sampling ratio for multiset
Rnode          : sub-sampling ratio for
                 local repository
n              : size of multiset
b              : burn-in period
sink           : initial node in random walk

Variables:
sample_size    : total size of data samples
                 before applying COLLAPSE
p2p_sample     : total sample from sampled
                 nodes
curr           : current node in random walk

Methods:
getNextMH(curr,b) {
   Get next node according to Metropolis-
   Hastings, using burn-in period b
}

run_estimator(sample, orig_size) {
   Estimates distinct values in sample
   with orig_size (size without running
   COLLAPSE)
}

Algorithm:

curr = sink;
while(sample_size < n * Rdata)
{
     curr = getNextMH(curr,b);
     sample(curr, Rnode);
     update sample_size;
     update p2p_sample;
}
run_estimator(p2p_sample, sample_size);
```

Figure 3: Pseudo-code for P2P network traversal

In Figure 4, we provide code for the sampling procedure at a node. The inputs have been described already in Figure 3.

```
Variables:
orig_sample    : original uniform-random
                 sample of data at a node
orig_size      : size of original sample
col_sample     : sample after running
                 COLLAPSE

Algorithm:

Sample(curr, Rnode) {
   orig_sample = uniform-random sample;
   orig_size = size of orig_sample;
   col_sample = collapsed orig_sample;
   return orig_size, col_sample;
}
```

Figure 4: Pseudo-code for sampling a node

Because random walks can be slow to yield results as they traverse the network, we can increase the response time by having multiple walkers running at the same time over the network. This can be a possible addition to our algorithm because assembling of node samples can be concurrently built from multiple sources.

4.5 Assumptions

Our algorithm operates upon various assumptions and constraints. We summarize these.

As with any random walk, we assume that tuples that match the query are frequent enough to be discovered by a random walk in order to perform distinct value estimation. We also assume that the data on the network does not change too much during the execution of the random walk.

Depending on the random walk stopping condition and the estimator, it may be required to know the total size of the network, or the total number of nodes. This may be available, assumed, or approximated in a preprocessing phase. This may introduce substantial errors into the system.

We also assume that the network cost of retrieving tuple attributes from sampled nodes is low. If this is not the case, more advanced stopping conditions such as the threshold technique can be used.

5. DISCUSSION

In this section we discuss the capabilities of our algorithm as well as the expected performance for different queries on varying P2P network topologies and applications. We show that the behavior of the algorithm can be customized to suit requirements by changing the values of the available input parameters.

5.1 Supporting Different P2P Network Characteristics

Since different topologies can form for unstructured P2P networks, it is important that our algorithm be able to handle a variety of clustering and connectivity scenarios that may arise.

As discussed previously, the use of the Metropolis-Hastings algorithm handles possible variations in the connectivity of nodes. Also, increasing of the burn-in period reduces the effects of node clustering. However, it is important to bear in mind that a larger burn-in period translates to longer wait times as the sampler converges to a uniform random sample.

It must be noted that the smaller the diameter of the P2P network, the faster a random sample is obtained [33]. This is intuitive since sampled peers can thus be further apart on opposite ends of the network (in terms of hops) and are less likely to be stuck in the same locality or cluster.

The authors in [33] also suggest possible modifications to the Metropolis-Hastings algorithm for attaining required node sampling distributions. They also show the suitability of the algorithm for several interesting applications and topologies.

Thus considering the design of the Metropolis-Hastings algorithm and the effect of changing the burn-in period, one can design the random walk to suit a variety of applications. Possible enhancements such as transition probabilities [33] can also give some nodes preference during the walk if this is required by applications.

Network costs associated with running the random walker also vary according to the type and scale of the network, the size of samples and various other factors.

As described earlier, our algorithm allows the node sampling ratio to be set for a walk. By tuning this value, one can set an appropriate sampling ratio that balances the cost of accessing nodes and the cost of retrieving samples from these nodes via the network. A smaller sampling ratio has the effect of producing smaller samples from nodes, reducing the time taken in transferring samples to the sink. However a consequence of this is that more samples need to be accessed from more nodes, and the random walk becomes longer. Conversely, for larger sampling ratios, the length and cost of accessing nodes in the random walk reduces, but because of the high sampling ratio, large-size nodes may have correspondingly large samples that are expensive to transfer to the sink.

Thus, depending on the specific application, the network, its scale, the distribution of node sizes, and the overall sampling ratio of the entire multi-set, it may thus be possible to design a walker with an optimal sampling ratio that minimizes overall network cost and wait times.

5.2 Supporting Different Data Distributions

Supporting distinct value estimation for different data distributions naturally depends heavily on the estimator that is used. Assuming we have an appropriately configured random walk that yields a uniform-random sample, we expect performance comparable to a random sample of nodes taken from the set of nodes in the system. We now analyze some of the parameters that model a data distribution, and assess how our algorithm will be affected for different types of distributions.

5.2.1 Effect of Clustering of Data

The amount of clustering of the data on the network changes the way estimation will be performed with our algorithm.

In perfect clustering, similar data is generally found within the same node or within the locality around the node. In low clustering cases, data is mixed and distributed across the network, resulting in less of a correlation between the data and its location on the network. The worst performance is expected for the first case, as each node produces a collapsed sample with a relatively small quantity of distinct values. Since the distinct values estimators use the original size of the sample before COLLAPSE, the estimator gains less information from the sample and the estimate has more error.

The performance of distinct value estimation using our algorithm increases as clustering reduces and duplicates are scattered more across the network. Samples from less clustered data, result in a sample that picks up more values (and their frequencies). This translates to a higher quality sample. Estimators scale these counts more accurately to produce a better approximation of the answer.

5.2.2 Effect of Data Skew

The skew specifies the shape of the frequency distribution curve. Highly skewed data has a small ratio of elements with very high frequencies in the multi-set, and a higher ratio of lower-frequency elements in the sample.

For high skew data, our random walk has a lower chance of sampling the rarer elements in the multi-set. The sample will instead have a lot of occurrences of high-frequency values that are duplicates across many nodes. This affects the quality of the sample and the distinct value estimator becomes more erroneous. Obviously, our algorithm handles data distributions with less skew well. These correspond to increasingly uniform distributions, where it is easier to get accurate samples.

6. EXPERIMENTATION

In this section, we provide experimental validation of our proposed approach. We have implemented the algorithms to run on both synthetic and real-world network topologies, with different network sizes, different data distributions and various clustering levels.

6.1 IMPLEMENTATION

Our algorithm was implemented in Java (Sun JRE 1.5.0) and carried out on Dual 3.00GHz Intel Xeon processors with 2GB RAM using the graph generation framework, JUNG [6]. The Java Runtime Environment was set to a maximum allowable heap size of 300MB.

6.1.1 Generation of P2P Networks

The P2P networks for running the algorithm are simulated by loading real-world topologies obtained from existing and highly popular unstructured peer-to-peer networks. The topologies that we loaded were collected from the Gnutella P2P network. They were obtained from actual data [37]. Our simulator loaded a total of 24278 peers with a total of 62391 edges in the system. A total of over 16.1 million integers were allocated to the nodes to run the simulations.

Each node is allotted a data set size that indicates the number of integers (tuples) stored at the node. These sizes ranged between 300 and 1500 tuples. The distribution of sizes is Zipfian [30] and several input parameters (described below) are used to control the shape and properties of the distribution. It is our intuition that this models the fact that different nodes have different sizes and that there are usually fewer nodes with very large databases and more nodes with smaller sizes.

6.1.2 Generation of Node Databases

We use single-attribute tuples for the node databases. A configurable number of distinct values (integers) are generated and duplicated so that they follow a Zipfian distribution. These numbers are stored in a sorted 'multi-set' which contains the entire data for the whole P2P system. Its total size depends on the parameters provided when generating node sizes (as described in section 6.1.1). The multi-set is then partitioned so that each node is allocated the correct number of tuples. This partitioning is done as the network is traversed using a breadth first search.

In order to simulate different levels of clustering – where the data in and around a node is similar – we shuffle the data in the multi-set before allocation. A highly shuffled multi-set has a lower level of clustering. This approach is adapted from [2].

The approach for generating data for the nodes follows the intuition that the distribution of distinct values, files follows Zipf's Law [5, 21, 30]. Some values tend to be very popular compared to others. Additionally, depending on the application, neighbors may share highly similar data. Each node may also contain mostly unique elements, with few duplicates residing in other nodes. Correlations such as these are modeled using the clustering technique described above.

6.2 Input Parameters

A variety of input parameters allow many different scenarios to be simulated to test our algorithm and its performance. These parameters affect the properties of the P2P network and the data that is distributed on the network. The operation of the algorithm can also be controlled with a separate set of parameters. They are described in the following section.

6.2.1 P2P Network Parameters

These parameters control the node sizes as they are allocated to all nodes in the graph. The actual data is generated and allocated to the nodes afterwards.

- **Node Data Size Skew (z_{size}):** This parameter controls the slant of the frequency distribution of the data sizes allotted to a node. The skew ranges between 0 and 1. Lower values indicate increasingly uniform (flat) Zipfian distributions.
- **Maximum and Minimum Node Data Sizes:** These sizes specify the highest and lowest possible number of tuples that are allocated to nodes on the network.
- **Number of Steps in Node Data Sizes:** The range of allowed data sizes is divided into evenly-spaced steps of increasing sizes, ranging from the minimum to the maximum allowable size. The step size is given by:

  (max_size – min_size)/num_steps

  Each node is allocated a data size at a step. Expressed simply, it is the distinct number of sizes generated within the range of permissible node sizes.

6.2.2 Node Database Generation Parameters

These parameters control the properties of the distribution of tuples on the entire P2P network. These tuples are modeled as integers in our implementation.

- **Number of Distinct Values:** This is the number of distinct data values that are distributed across the network.
- **Data Distribution Skew (z_{data}):** The distinct values are duplicated to follow a Zipfian frequency distribution. The skew value controls the shape of this distribution. The number ranges from 0 to 1. Lower values correspond to increasingly flat (uniform) distributions. Larger values have distributions that slant more steeply since a few elements will be duplicated far more than others.
- **Cluster Level (C):** The Cluster level indicates the level of mixing that is performed on the sorted data before it is allocated to the nodes on the P2P network. This value ranges from 0 to 1. A Cluster Level of 0 performs no mixing, and this corresponds to a perfectly clustered network, where each node and its neighbors hold very similar data. As the clustering level tends towards 1, the data is mixed entirely before it is spread onto the network.

6.2.3 Algorithm Input Parameters

The following parameters control the way that our algorithm runs.

- **Burn-In Period (B):** This is the Metropolis-Hastings algorithm parameter that controls the number of times that the coin is tossed before a node is sampled. The coin toss decides whether to move to a node or not. Thus, successive sampled nodes on the random walk may be between 0 and B hops apart based on the outcomes of these decisions.

- **Node Sub-Sampling Ratio (r_{node}):** This is the sampling ratio for the local repository of a node. It allows the sample size from a node to be larger for nodes that contain more tuples.
- **Data Sampling Ratio (r_{data}):** This is the overall sampling ratio of the data on the entire network. It can provide a stopping condition for the random walk by keeping track of the total sample size as samples are accumulated during the walk.
- **Threshold (T):** The threshold can be set as a stopping condition for a random walk. It requires the definition of a function that models the cost of accessing and retrieving samples from nodes on the walk.

6.3 Evaluation Metrics

Evaluation of the algorithms can be done by assessing the ratio error of the estimates. The ideal estimator has a ratio error of 1.

Other evaluation metrics could include the number of node samples, the total number of samples collected and the number of messages exchanged.

6.4 Experiments and Results

We compare the results of running a random walk (while using Metropolis-Hastings) with burn-in period of 10 with a simple random node sample from of the set of nodes in the system. This helps in assessing the quality of our traversal method.

We also compare with a uniform random sample of the data, taken directly from the set of all numbers in the multi-set. This helps us to assess the estimators to see how close we can get our algorithm to match performance on a uniform-random sample.

All tests are carried out by averaging the results from 10 runs of the algorithm.

6.4.1 Node Sub-Sample Ratio

Figure 5 shows the effect of changing the sub-sampling ratio at a node. The overall multi-set data sampling ratio (Rdata) is set at 3%, with a skew parameter of 0.5 and a clustering level (Cl) of 0.7. The number of distinct elements on the system is held at 500,000 and we use AE to estimate the number of distinct values using different sampling ratios.

We observe that at lower data sample ratios, the error of the random walk and the random node sample methods approach that of the random data sample from the multi-set.

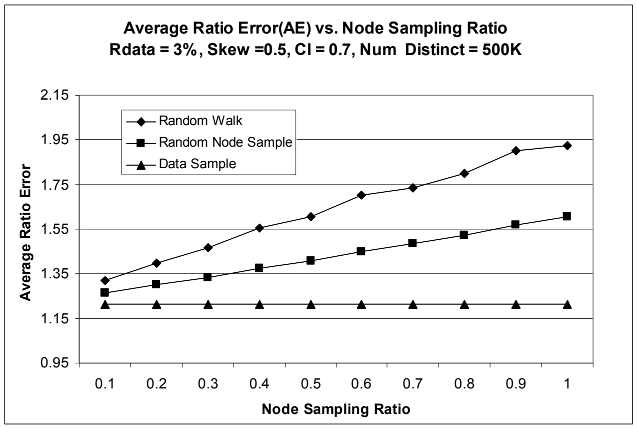

Figure 5: Average Ratio Error vs. Node Sampling Ratio

6.4.2 Data Sample Ratio

Figure 6 shows the effect of varying the overall data sampling ratio for the multi-set. As expected at lower values (0.5%), the ratio-error is very high. It improves significantly and all three sampling strategies have comparable results with the sampling ratio greater than 1.5%.

The result is obtained with a node sub-sampling ratio (Rnode) of 0.5.

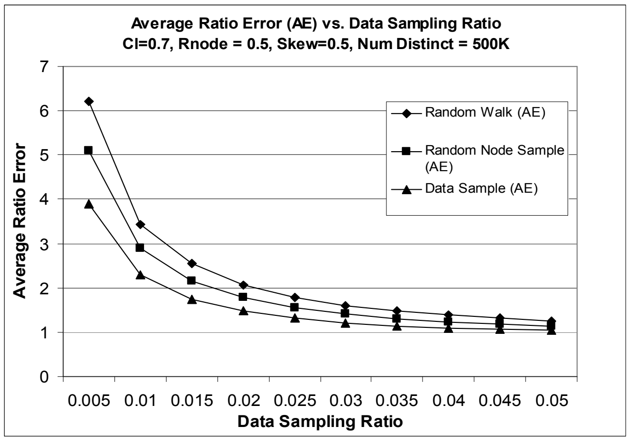

Figure 6: Average Ratio Error vs. Data Sampling Ratio

6.4.3 Clustering Level

Figure 7 shows the effect of varying the clustering level of the data. A cluster level of 0 indicates perfectly clustered data and the expected high ratio error can be observed. The quality of the estimation improves significantly very rapidly. This demonstrates the importance of clustering in our sampling scheme.

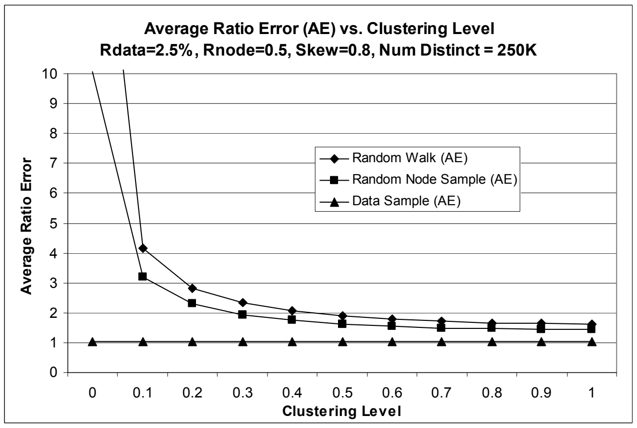

Figure 7: Average Ratio Error vs. Clustering Level

6.4.4 *Data Skew*

To demonstrate the performance of the algorithm with different distributions, we vary the skew of the data as shown in Figure 8. This shows how, for higher skews beyond 0.8, the ratio-error increases greatly.

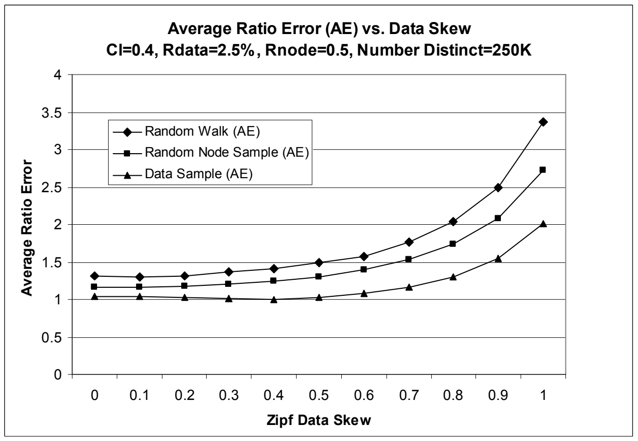

Figure 8: Average Ratio Error vs. Zipf Data Skew

6.4.5 *Number of Distinct Values*

We provide two graphs here to demonstrate the operation of both the GE estimator and AE.

The AE graph shown in Figure 9 is obtained by varying the number of distinct values on the network between 100,000 and 1,000,000.

Figure 10 shows the graph obtained for the GE estimator, while varying the number of distinct values between 10,000 and 100,000. Note that far fewer distinct elements were used for estimations using GEE as it produces poor results at a higher number of distinct results. This is expected, since GEE is a biased, optimal error estimator.

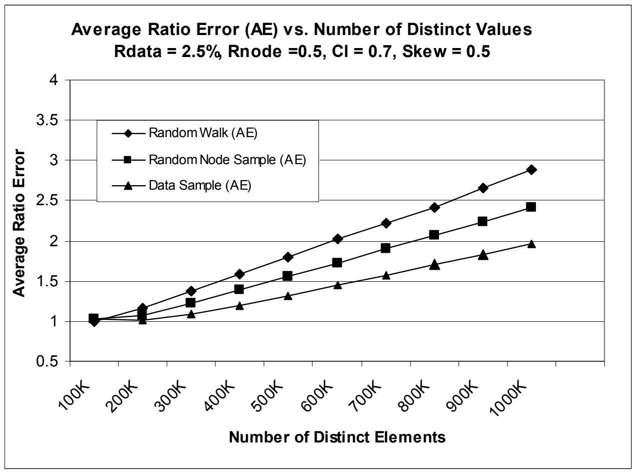

Figure 9: Average Ratio Error vs. Number of Distinct Elements using the Adaptive Estimator

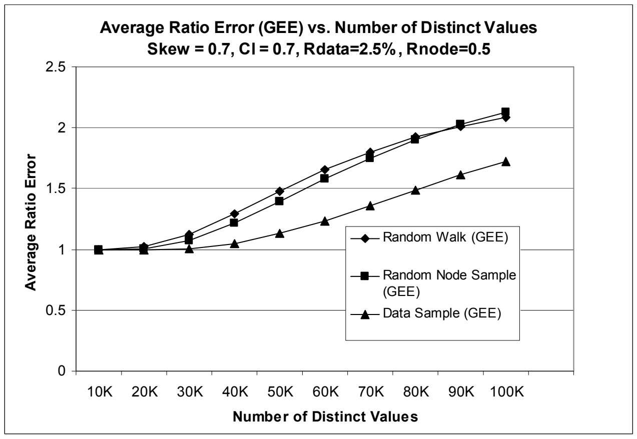

Figure 10: Average Ratio Error vs. Number of Distinct Elements using the Guaranteed Error Estimator

6.4.6 *Burn-In Period*

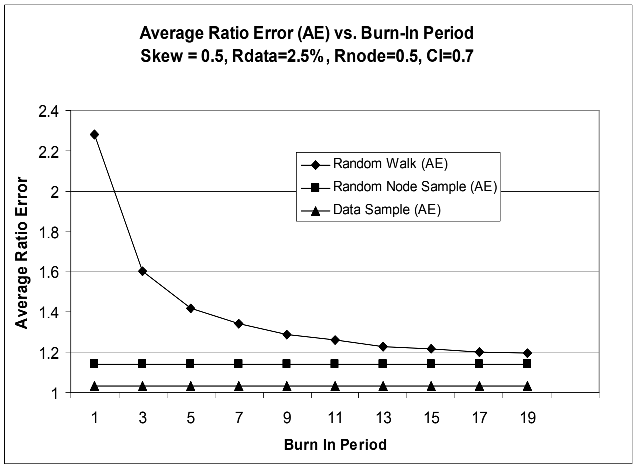

Figure 11: Average Ratio Error vs. Burn-In Period

Figure 11 shows the effect of executing the random walk with different burn-in periods set. As seen from the results, the ratio-error becomes increasingly close to that of a random sample as the burn-in increases. This is because the sampled nodes become increasingly less correlated as the burn-in period increases.

7. CONCLUSION AND FUTURE WORK

In this paper we address a new and interesting problem with unique challenges. We offer a way of combining different strategies to come up with an algorithm that has a ratio error comparable to uniform-random node samples, and in some conditions, even comparable to uniform-data samples. It also opens up numerous possibilities for future work.

Distinct value estimation for dynamic graphs could be studied for P2P network environments which change rapidly during the execution of a random walk.

Similarly, ways of predicting how much to sample from the P2P network could warrant further study. The challenges of this are well known, considering the difficulty in predicting error in distinct value estimates.

Another area for future work is to model the network costs of accessing and retrieving samples from nodes. The ability to do this will allow optimization of input parameters for the algorithm, and will also offer a means of predicting when to bound the random walk.

Being able to get distinct values estimations also paves the way for future work in histogram construction and query optimization on peer-to-peer networks. This is could be an emerging area due to the rapid increase in popularity of P2P networks and the possible paradigm shift away from traditional client-server models.

8. REFERENCES

[1] Chaudhuri, S., Das, G., and Srivastava, U. Effective Use of Block-Level Sampling in Statistics Estimation. *SIGMOD 2004* (Paris, France, June 13-18, 2004)

[2] Arai, B., Das, G., Gunopulos, D., and Kalogeraki, V. Approximating Aggregation Queries in Peer-to-Peer Networks. *ICDE 2006* (April 3-8, Atlanta, GA, 2006)

[3] Bar-Yossef, Z. and Gurevich, M. Random Sampling from a Search Engine's Index. *International World Wide Web Conference Committee 2006* (Edinburgh, Scotland, May 23-26, 2006).

[4] Charikar, M., Chaudhuri, S., Motwani, R., and Narasayya, V. Towards Estimation Error Guarantees for Distinct Values. In *Proceedings of the ACM Symposium on the Principles of Database Systems, 2000.*

[5] Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. Search and Replication in Unstructured Peer-to-Peer Networks, *ICS 2002*, (June 22-26, 2002, New York, New York, USA)

[6] Java Universal Network/Graph Framework (JUNG) Website. http://jung.sourceforge.net.

[7] Metropolis, N., Rosenbluth A,, Rosenbluth, M., Teller, A., and Teller,E.. *Equations of state calculations by fast computing machines*. J. of Chemical Physics, 21:1087–1091, 1953.

[8] W. Hastings. *Monte Carlo sampling methods using Markov chains and their applications*. Biometrika, 57(1):97–109, 1970.

[9] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. *In Proceedings of International Conference on Peer-to-peer Computing*, August 2001.

[10] Gnutella Development website: rfc-gnutella.sourceforge.net

[11] Stutzbach, D., Rejaie, R., Duffield N., Sen, S., and Willinger, W. On Unbiased Sampling for Unstructured Peer-to-Peer Networks. *Internet Measurement Conference (IMC) 2006* (October 25–27, 2006, Rio de Janeiro, Brazil)

[12] Baset, S.A., and Schulzrinne, H. *An analysis of the Skype peer-to-peer Internet telephony protocol*. Technical Report CUCS-039-04, Computer Science Department, Columbia University, September 2004.

[13] KazaA website: kazaa.com

[14] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *Proc. of the 1998 ACM SIGMOD*, pages 436–447, 1998.

[15] P. Haas, J. Naughton, P. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. of the 1995 Intl. Conf. on Very Large Data Bases*, pages 311–322, Sept. 1995.

[16] Tsoumakos, D., and Roussopoulos, N. - A Comparison of Peer-to-Peer Search Methods In *Proceedings of the Sixth International Workshop on Web and Databases,* San Diego, California (June 12-13, 2003)

[17] Androutsellis-Theotokis, S., and Spinellis, D. *A Survey of Peer-to-Peer Content Distribution Technologies*. ACM Computing Surveys, 36(4):335-371, December 2004.

[18] Bar-Yossef, Z., Berg, A., Chien, S., Fakcharoenphol, J., and Weitz, D. Approximating Aggregate Queries about Web Pages via Random Walks. In *Proceedings of the 26th International Conference on Very Large Data Bases* (September 10 - 14, 2000).

[19] King, V. and Saia, J. 2004. Choosing a random peer. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing* (St. John's, Newfoundland, Canada, July 25 - 28, 2004)

[20] Kempe, D., Dobra, A., and Gehrke, J. 2003. Gossip-Based Computation of Aggregate Information. In Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (October 11 - 14, 2003).

[21] Ganesan, P., Bawa, M., Garcia-Molina, H. Online balancing of range-partitioned data with applications to peer-to-peer systems .In *Conference on Very Large Databases (VLDB) 2004* (Toronto, Canada, August 31 - September 3 2004)

[22] Hou, W., Ozsoyoglu, G., and Dogdu, E. Error-Constrained COUNT Query Evaluation in Relational Databases. In *Proc of the 1991 ACM SIGMOD*, pages 278–287, 1991.

[23] Hou, W., Ozsoyoglu, G., and Taneja, B.. Statistical estimators for relational algebra expressions. In *Proc. of the 1988 ACM Symp. on Principles of Database Systems*, pages 276–287, Mar 1988.

[24] Gkantsidis, C., Mihail, M., and Saberi, A. Random Walks in Peer-to-Peer Networks. In *INFOCOM, 2004*

[25] Awan, A., Ferreira, R.A., Jagannathan, S., and Grama, A. Distributed Uniform Sampling in Unstructured Peer-to-Peer Networks. In *Hawaii International Conference on System Sciences, 2006.*

[26] Ntarmos, N., Triantafillou, P., Weikum, G., Counting at Large: Efficient Cardinality Estimation in Internet-Scale Data Networks, In the *22nd International Conference on Data Engineering (ICDE'06), 2006.*

[27] L. Goodman. *On the estimation of the number of classes in a population.* Annals of Math. Stat., 20:572–579, 1949.

[28] K. Burnham and W. Overton. *Robust estimation of population size when capture probabilities vary among animals*. Ecology, 60:927–936, 1979.

[29] Shlosser A. *On estimation of the size of the dictionary of a long text on the basis of a sample*. Engrg. Cybernetics, 19:97–102, 1981.

[30] Zipf, G. E. *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press, Inc., 1949.

[31] Lipton, R., Naughton, J., and Schneider D. Practical selectivity estimation through adaptive sampling. In *Proc. Of the 1990 ACM SIGMOD*, pages 1–11, 1990.

[32] Haas, P. and Swami, A. Sequential sampling procedures for query size estimation. *In Proc. of the 1992 ACM SIGMOD*, pages 341–350, 1992.

[33] Zhong, M. and Shen, K. *Random walk based node sampling in self-organizing networks*. SIGOPS Oper. Syst. Rev. 40, 3 (Jul. 2006), 49-55.

[34] Le Fessant, F., Handurukande, S., Kermarrec, A.-M., and Massoulié, L. Clustering in Peer-to-Peer File Sharing Workloads. *3rd Intl. Workshop on Peer-to-Peer Systems IPTPS 2004*

[35] Babcock, B, Chaudhuri, S., and Das, G. Dynamic Sample Selection for Approximate Query Processing. *SIGMOD Conference 2003*: 539-550.

[36] Chaudhuri, S., Das, G., Datar, M., Motwani, R., and Narasayya, V.. Overcoming Limitations of Sampling for Aggregation Queries. *ICDE 2001*: 534-542.

[37] Details omitted due to double-blind reviewing.