DISTINCT VALUE ESTIMATION BY SAMPLING ON UNSTRUCTURED PEER TO PEER NETWORKS

by

ZUBIN MATTHEW JOSEPH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE & ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2007

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Gautam Das, who guided, motivated and encouraged me while working on this thesis. His patience and understanding are valued greatly along with all the time and effort spent in discussing and analyzing the challenges that this research problem has posed.

I would also like to express my gratitude to my defense committee, Dr. Nan Zhang and Dr. Leonidas Fegaras, who also played a major part in this work right from the inception of this research problem.

I'd also like to thank Benjamin Arai for the tips that helped me successfully run the experiments and simulations.

I'd also like to thank my family, my friends and my trusty laptop for always being there for me. I also owe a special thank you to everyone from the DBX lab for their constant support.

November 8, 2007

ABSTRACT

DISTINCT VALUE ESTIMATION BY SAMPLING ON UNSTRUCTURED PEER TO PEER NETWORKS

Publication No. ______

Zubin Matthew Joseph, MS

The University of Texas at Arlington, 2007

Supervising Professor: Gautam Das

Peer-to-Peer networks have become very popular on the Internet, with millions of peers all over the world sharing large volumes of data. The sheer scale of these networks has made it difficult to gather statistics that could be used for building new features. This thesis presents a technique of obtaining estimations of the number of distinct values matching a query on the network. The method is then analyzed by considering simulation results that demonstrate its effectiveness and flexibility in supporting a variety of queries and applications.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

Peer-to-Peer (P2P) networks are highly popular medium for sharing CPU processing power, storage space, and/or content in the form of text documents and various forms of media. Popular networks such as Gnutella [17] and KazaA [25] often share music files, while applications such as Skype [7] even use P2P networks for Voice over IP (VoIP) telephony.

These networks usually operate over the Internet and consist of thousands, and millions of peers that can be located anywhere in the world. 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 thesis, peers do not make any assumptions about the location of other peers, the distribution of 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. All peers are considered ‘equal’, in that they can be either servers or clients, depending on the services or resources that they provide or access from other peers. With the vast number of participating peers, all of which are free to join and leave the network at any time, it is difficult to keep track and gather statistics on the large volumes of the data available on the P2P network. This thesis focusses 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. This is a well known problem in the domain of databases and it is especially important for query optimization, especially in the construction of histograms [9, 10, 12, 20]. 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

Having distinct value (DV) estimation techniques available for P2P networks would not only allow histogram construction in the future, but would also enable management, administration, monitoring, and report generation features to be built into such systems.

The capability of obtaining estimations to answer queries placed at any node in the network allows trends in popularity to be gathered by making use of density and duplication values. This could be extended to top-k ranking, and even for estimating term frequencies and statistics on keywords in documents in a P2P database. Consider an IR application where each document is considered as a node. For a search term X, it is possible to estimate the total number of distinct words in the entire system that appear in the same document as X. Our algorithm is even more applicable in this case if the documents can have links to each other, as with the web. Having knowledge of a value like this can help to assess popularity, importance and relevancy of a word. For example, common terms will have a high number of distinct values, but more specific words will have a lower number of distinct words that they are associated with, indicating their relevancy.

Other new and exciting applications become possible. Consider a peer-to-peer (P2P) 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 can be used to assess overall trends in queries. Furthermore, the technique can be used to assess IP and access logs across peers so that patterns in network or user behavior and can be deduced.

1.2 Challenges

Distinct value estimation on unstructured P2P networks is a new problem that has not been investigated yet to the best of our knowledge.

We view the P2P network as a large database table that is conceptually partitioned (with overlap) and distributed across a large number of peers. Answering distinct value queries on such a system is challenging because we aim to answer queries executed on unstructured P2P networks, where each peer only knows information about the location of its neighbors. Without any knowledge of the network, the naïve approach to answering distinct value queries is to query each peer in the system by flooding the entire P2P network. Each peer individually sends all its distinct tuples to the query initiator (sink). The sink then performs a union of all the tuples from all the peers and then eliminates duplicates. The size of this set is the number of distinct values on the network. Although this approach is exact, it is obviously prohibitively expensive in terms of time, space and the associated network costs of accessing and retrieving data from all the nodes on the network.

A variety of approximate techniques such as sketches [13] can be used to make operations such as these more space efficient. However, accessing all the data on all the nodes on the network would still take too much time. It is thus evident that any solution to this problem would need to allow sampling a small subset of the data on such networks. One of the most apparent difficulties with this is that distinct value estimation in a centralized data repository itself is known to be a hard problem, as proved in [9]. A variety of estimators [9, 12, 20] exist, but currently none provide guaranteed error bounds for a uniform-random sample of tuples in the column of a table [10].

The potential clustering of similar data at peers adds another interesting dimension to the problem. Clustering can arise because sharing and duplication of data is more likely to occur between neighboring or close-proximity nodes. For example, nodes located in France may share data that is only popular in France. With such constraints, it is a challenge for any peer on the network to obtain a uniform random sample so that a distinct value estimator can be applied. Furthermore, the calculations for gathering such estimates may require parameters that are unavailable to a peer, such as the total size of the data available on the network.

1.3 Contributions

This thesis presents an algorithm for distinct value estimation on P2P networks. It is effective because:

• It does not require that peers exchange calculations or global constants governing algorithm behavior.

• It reduces preprocessing and needs minimal knowledge of the P2P network properties.

• It minimizes the information that a node needs to maintain about its neighbors.

• It is largely independent of the connectivity, clustering and/or data distribution characteristics of the P2P network.

• It allows the flexibility of changing distinct value estimators.

• The quality of estimates, in some cases, approaches that of a uniform random sample of the entire dataset of the network.

The rest of this thesis is organized as follows. In Section 2 we discuss related work. In Section 3 we describe our algorithm for computing distinct values over P2P databases. In Section 4 we discuss how our algorithm can be applied for answering queries on a P2P network with different data distribution properties. In Section 5 we describe a comprehensive set of experiments that demonstrate the effectiveness of our approach. We conclude in Section 6.

CHAPTER 2

RELATED WORK

A variety of search and node traversal techniques for both structured and unstructured peer-to-peer networks are surveyed and described in great detail in [1, 36].

Extensive studies have been done on random walks including [16, 30]. The Metropolis-Hastings algorithm is discussed [5], and is used to execute a random walk over documents indexed by a search engine. Alternatives such as the Random Weight Distribution method are suggested in [3] but this requires support from the underlying P2P network. [35] suggests a modification to the Metropolis-Hastings sampler that makes it suitable for dynamic graphs. In this thesis however, 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 in [19, 22, 23, 29]. Sampling has also been used for approximate query processing in centralized databases [4, 9, 10, 11, 12, 16].

Additionally, several authors have looked into approximation-type queries for P2P networks, including using random walks over the web in [6], and aggregations over unstructured P2P networks as in [2]. Alternative gossip-style techniques of computing aggregates have been suggested by [26], but require participation of every node in the system. Techniques utilizing structured P2P networks have addressed the problem of sampling random peers [27], approximations [32]. There is a large body of work on these types of networks but we target unstructured P2P networks, which require a different approach.

In statistical literature, cluster sampling is a concept considered similar to block-level sampling [10]. In this thesis we consider this technique of sampling, originally proposed in [22, 23]. An analysis of block-level sampling on databases is discussed in [10], where block-level estimates are used to construct histograms and perform distinct value estimation. The authors in [12] consider this and also suggest an optimal error distinct value estimator. Other Distinct Value Estimators have been proposed, such as the Adaptive Estimator [9], the Goodman Estimator [18] and other estimators in [8, 20, 34]. The authors in [13] use probabilistic methods to count the number of distinct elements in a set of data. The technique involves constructing sketches that can be combined from multiple sources. This is well suited to large databases and data streams. The use of sketches requires a pass over the entire dataset in order to build the sketch. Thus they are not practical in the P2P scenario, where all the data elements at every node on the network would need to be accessed to construct and maintain a sketch. The authors in [15] proposes maintaining a distinct sample that can be used for distinct value estimates, but this also has the drawback of requiring at least one pass over all the data on the network. Additionally, the techniques in [13, 15] cannot be combined with a random walk as they produce an estimate of distinct values already encountered during the walk. The techniques cannot in any way factor in an estimate of the number of distinct values at nodes that are not selected during the walk.

CHAPTER 3

DISTINCT VALUE ESTIMATOR FOR P2P NETWORKS

We first provide the foundations of our approach to solving this novel problem. We then discuss and provide a concise definition of our algorithm.

In this thesis, 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 thesis. We assume that each peer Pi has a local database Di, and refer to the total data D on the network including duplicates, as a multi-set as it is the union of all the sets of data residing at each peer in the network. For example, on Gnutella each peer stores a local database of songs, and the data stored on the entire network may be viewed as a multi-set due to the existence of duplicates. We consider SQL-like queries of the form "SELECT COUNT (DISTINCT *) FROM D WHERE <selection condition>". 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 first consider two distinct value estimators, the Guaranteed-Error Estimator (GEE) [12] and the Adaptive Estimator [9]. 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) [9, 12] is an estimator with optimal error and a bias of at most $O(\sqrt{1/q})$ [9, 10, 12].

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

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) [9] is a heuristic distinct value estimator. It takes on the form:

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

K is an appropriate scaling factor that is computed from the sample and d is the number of distinct values in the sample, given by:

$$d = \sum_{i=1}^{r} f_i$$

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 sample. Multiple occurrences of an element are not scaled. AE, unlike GEE, does not require a value for n (the total size of the multi-set). However, both estimators require a uniform-random sample and the counts (f_i values) of the elements that occur i times in a sample of r elements. We now discuss techniques of obtaining such a sample from an unstructured P2P network.

3.2 Network Traversal

We first discuss how we can traverse the P2P network in order to obtain a random sample of nodes to sample from. In this section, we review a few approaches for traversing an unstructured P2P network, where no peer has any knowledge of the rest of the network, except for network connections to a small subset of neighbors.

3.2.1. Flooding

The obvious method of sampling the set of peers is to use *flooding*, where the query initiator sends out messages to all its neighbors. These in turn pass the message on to their neighbors till the message spreads throughout the network to every node. It places great load on participants and on the network [30] due to repeated messages and cycles.

3.2.2. Random Walks

A popular technique is to use a *random walk*, where each node picks out only a random neighbor (or subset) to pass each message it receives. Although this process has a high latency, it allows a chain of nodes to be gathered as messages pass from one neighbor to the next. Random Walks are far less resource intensive than more exhaustive techniques such as flooding the network, where all the nodes in the system are accessed. The random walk offers a good way for us to collect a sample of random nodes from the network.

3.2.2.1 Random Walks and Clusters

In real-world peer-to-peer networks it is inevitable that nodes form clusters. These clusters may consist of peers that have similar data residing on the network. Alternatively, these clusters may form because the nodes may all be in the same geographical area or because of characteristics of the P2P network. If clusters form for any reason on the network and there is a small cut between clusters, then a random walk has a high chance of getting stuck repeatedly sampling nodes from within a cluster, rather than from across the entire network. In [2], the effect of this clustering is reduced by setting a jump size *j* for the walk, as shown in Figure 1. Only nodes that occur after *j=2* hops are taken into the sample. Setting larger jump sizes during a walk also has the advantage of reducing the correlation between successively sampled peers.

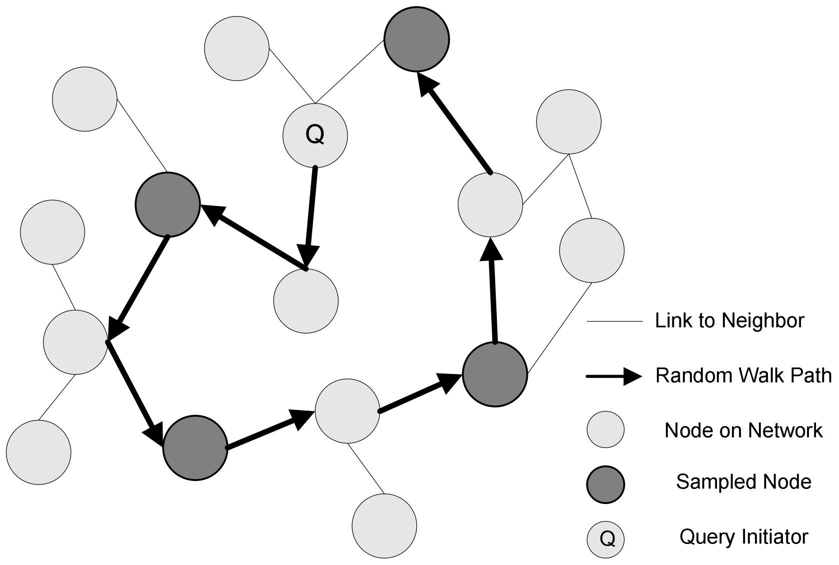

Figure 1: Random Walker with jump size = 2

3.2.2.2 Random Walks and Varying Node Degree

Different nodes in the network may have a different number of neighbors. In Gnutella, some peers may collect many neighbors (called ultrapeers) [1, 36]. 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 visiting a node that has a large degree, over one that has a smaller degree. This results in a non-uniform sample with a higher chance of duplicates of such nodes. Since a random walk has a higher chance of picking out well-connected nodes for sampling in this manner, the varying degree of nodes also poses a problem.

The effect of this can be reduced by using the Metropolis-Hastings (MH) algorithm [5]. This changes the way a node chooses a neighbor to move to during the random walk process. If *deg*(*X*) is defined as the total number of neighbors (edges) of node X, the MH algorithm defines an *acceptance probability* *r*(*X,Y*):

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

Consider a Node *X* that randomly chooses a neighbor *Y* as the next node in the random walk. *X* then performs a *coin-toss*, where the walk moves to *Y* with probability of *r*(*X,Y*), otherwise it stays at the same peer and picks out another neighbor. Thus, the path of the walker is governed by the acceptance probability, which is such that a higher preference is given to moving towards nodes with lower degree as there is a higher chance of accepting a move to such a node. The result is a sample of nodes that are not necessarily selected because they have a higher degree. It results in random walks that are more random-uniform since they are less dependent on degree.

Just as the technique described in [2] samples nodes after every *j* hops, the MH algorithm uses the *burn-in period b* and samples the current node after *b* coin-tosses during the walk (see [5, 21, 31]). Depending on the result of each coin toss, the number of actual accepted hops between sampled nodes ranges from 0 to *b*. Setting a higher burn-in period makes it less likely that the random walker will get stuck sampling nodes within a cluster and also reduces the correlation between successively sampled nodes.

The Metropolis-Hasting method of traversal thus allows us to obtain a random-uniform sample of peers from the network and even allows us to flexibly control the sampling using parameters such as the burn-in. We now discuss how to sample the local databases at each node selected for sampling via a Metropolis Hastings random walk. We also suggest how to process these samples for DV estimation.

3.3 Block-Level Sampling on P2P Networks

By viewing the local databases of peers as blocks in a traditional database, we propose a technique of applying block level sampling methods [10] to nodes on P2P networks. While this technique is originally applied to centralized databases, it offers a variety of benefits that are applicable to P2P networks.

As shown in Figure 2, we sample the local databases of peers chosen for sampling using the MH random walker. These samples are processed by the COLLAPSE algorithm by removing all the duplicates from records that pass the query selection condition at a peer. The resulting duplicate-free dataset is sent over the network to the query initiator (sink). The sink forms the union of all the sets it receives. The resulting set is then used with a suitable DV estimator to produce estimations for the entire P2P network.

Thus the resulting set contains duplicates from across peers, but no duplicates from within the same peer. This is the sample that is considered to be a sample of the entire data on the network. Once the frequencies of the elements in the sample are counted, the DV estimators normally scale only the number of single occurrences in order to come up with an estimate for the total number of distinct values in the entire database.

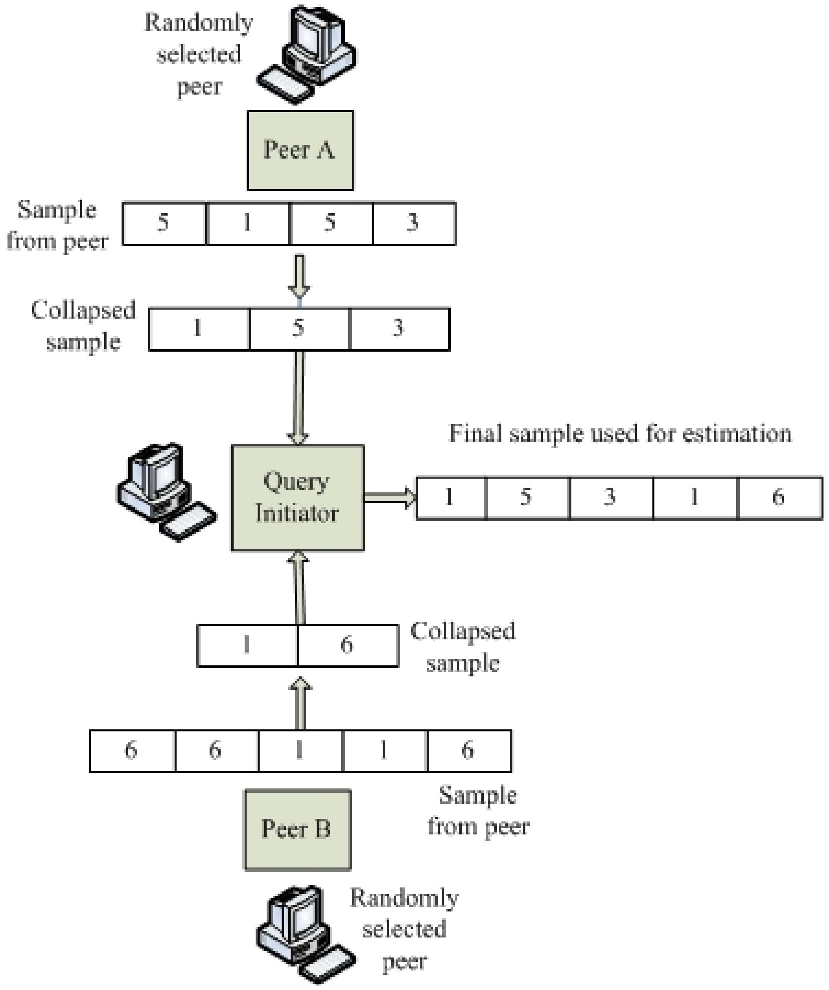

Figure 2: COLLAPSE on P2P Networks

The original block-level sample size is maintained by the algorithm for keeping track of the overall sampling ratio. Thus, the value for the total sample size that we use for distinct value estimation must be this original pre-collapsed size of the sample, and not the size of the duplicate-free sample [10].

Because peers on a P2P network are autonomous, they can enter, leave and add or remove shared resources at any time. Thus any peer can potentially be an adversary by inserting duplicates or multiple tuples that match a query on a particular attribute. An algorithm that can handle this susceptibility is crucial. Our adaptation of the original COLLAPSE algorithm addresses this concern as it can handle such adversarial models [10] because it factors out the duplications within a block and only deals with duplication of data across blocks. Furthermore, because data within a peer can be highly correlated, the duplicate-removal step makes the algorithm less vulnerable to repeatedly counting similar tuples from within a peer.

The block-level sampling technique is a simple algorithm that requires few additional input parameters. Since it makes no assumption on the size of each block, it can be used on different-sized blocks, making it suitable to be applied to P2P networks where nodes have local repositories of varying sizes.

3.4 Sub-Sampling Strategies at Nodes

Because peers can have databases of any size, at some nodes we may have too many tuples to send across the network to the query initiator (sink), unless some sort of compression/hashing technique is used. Thus, it becomes necessary to obtain a uniform random sub-sample from each sampled node's local repository. It is not possible to set a constant size for the sample from each node 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.

Although it may be possible to use different sampling ratios at peers depending on their degrees, we instead opt for a simple sampling technique that uses a constant sub-sampling ratio for nodes sampled during the random walk. This ensures that all the tuples residing on a sampled node are sampled by the algorithm in a fair manner.

3.5 Implementation Issues

We consider some of the implementation issues that must be considered in order to run the algorithm.

One of the first issues to consider is that the random walker must be designed carefully so that it has a stopping condition; otherwise it would continue executing infinitely. A straightforward solution is to execute the walk until the total accumulated sample size reaches a specified ratio of the total data size on the network. If we choose to use an estimator such as GEE (3.1.1), we can assume that a value for n is already available as it is also required for estimation. This value must either be estimated from a preprocessing step or already be known for a particular network, query or application. The only potential issue of using this technique of bounding the walk is that if large-sized peers are sampled, then the total required sample size might be reached quickly and sampling will not occur across a large number of peers. This affects the quality of samples as the more peers are sampled, the higher the quality of the final sample. This would introduce an additional error in our estimations. Furthermore, if all the peers are small, it may result in a walk that is very long as it samples more peers. This would result in longer wait times in order to get results.

Alternatively, the stopping condition can be set to sample a ratio of the nodes on the P2P network. Since nodes can have databases of different sizes, the technique may also be prone to error as there is no way to be sure of how much data is being sampled during the walk. This can lead to poor samples if the walk only samples smaller peers. Additionally, the number of peers on the network may also need to be assumed.

We now consider a possible method of providing a bound to the random walk by controlling the overall data sampling ratio and the node sampling ratios. We refer to it as the threshold technique.

3.5.1. Threshold Technique

The threshold operates by assuming that we have a cost function combining the time and network costs of reaching a sampled node (via the random walk) and the corresponding costs for sending back 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 cost T at which the walk should terminate, i.e., so that once the cost exceeds the threshold, the walk stops. This approach allows the additional flexibility of setting the walker to sample nodes by controlling of the total length of the walk and the amount of data sampled and sent back to the sink from each sampled node.

Thus, a lower sub-sampling ratio results in a longer walk that samples more nodes but a larger sub-sampling ratio samples fewer nodes but on a shorter walk that yields lower quality estimates. Similarly, the estimate improves as the overall data sampling ratio increases. However, as the ratio increases, the cost in sending back samples also increases. Thus, depending on the threshold and the modeled network characteristics, it should be possible to choose both of these ratios so as to get the best possible results for a query within a defined threshold time/cost.

A further investigation of this technique and the possibility of dynamically choosing sub-sampling ratios for sampling nodes is left for future work.

3.6 Pseudo-Code

We provide simple pseudo-code that defines our algorithm. We first provide the code for the random walk sampler. The pseudo-code makes calls to COLLAPSE [10], the Metropolis-Hastings sampler [5] and 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 for distinct value 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
                  at a node
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:

1:   curr = sink;
2:   while(sample_size < n * Rdata)
3:   {
4:        curr = getNextMH(curr,b);
5:        sample(curr, Rnode);
6:        update sample_size;
7:        update p2p_sample;
8:   }
9:   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.

3.7 Assumptions

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

Firstly, we assume that tuples that match the query are not so rare that a random walk fails to find any nodes to sample. We also assume that the data on the network does not change drastically during the walk.

Depending on the random walk stopping condition and the estimator, the algorithm may need the total number of tuples or nodes on the network or the total original sample size. This may be available, assumed, or approximated in a preprocessing phase. As we shall see in the experimentation section, the AE estimator performs well and does not require any such information. In cases where a parameter describing the network is required, many P2P networks clients (such as in Gnutella) have easy access to roughly how many peers are connected to the network at a given time. This allows a stopping condition to be set for the walk.

We also assume that the network cost of retrieving tuple attributes from sampled nodes is low. This is a fair assumption since even in P2P networks that are used for exchanging multimedia files (such as MP3 files that range between 3-7MB), it is sufficient to transfer only ID3 tag information and/or file hashes to identify duplicate files, as in [25]. These descriptors are usually text and are only a few kilobytes in size. If this is not the case and tuple attributes are too expensive to transfer to the query initiator, more advanced stopping conditions such as the threshold technique can be used as described in 3.5.1.

CHAPTER 4

DISCUSSION

In this section we discuss the behavior of the algorithm and how it can be adapted for different queries and applications by changing the values of the available input parameters.

4.1 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 topological scenarios that may arise.

As discussed previously, the use of the Metropolis-Hastings algorithm handles possible variations in the degree of nodes. Also, increasing of the burn-in period reduces the effects of node clustering. It is important to bear in mind that a larger burn-in period translates to longer wait times though the sample becomes closer to a uniform random sample.

A random sample is obtained much faster if the diameter of the P2P network is small [37]. This is intuitive since sampled peers can 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 [37] 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. Enhancements such as transition probabilities [37] can also give some nodes preference during the walk if this is required by applications.

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.

4.2 Supporting Different Data Distributions

Different P2P networks can be considered to form graphs that have different data distributions across the nodes. The data distributions can generally be described by the amount of skew in the data on the network, the amount of clustering/mixing of data across nodes, as well as the variation of sizes of peers on the network. Our algorithm is able to handle a variety of distributions with different characteristics.

4.2.1. Effect of Distribution of Node Sizes

Because peers on a network can have local repositories with different sizes, this affects the way samples are gathered from nodes. If all the peers generally have small databases, it may sufficient to set high sampling ratios and vice versa.

The fact that the node sizes can lie in a large range of possible values makes sampling at nodes and the choice of sampling ratio an interesting problem. As described earlier, our algorithm allows the sub-sampling ratio for a node's local database to be set for each a walk. By tuning this value, one can set an appropriate sampling ratio that balances the cost of reaching/accessing nodes and the cost of retrieving samples from these nodes via the network. A smaller sampling ratio produces smaller samples from nodes, reducing the time taken in transferring samples to the sink. A consequence however, 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 is possible to design a walker with a sampling ratio that strikes the right balance between estimation quality and overall network cost and wait times.

4.2.2. Effect of Clustering of Data

The clustering level of data on the network changes the performance of the 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 value estimators use the original size of the sample before

COLLAPSE, the estimator often gets fewer actual samples from highly clustered node after collapsing. Also, with higher clustering the duplicates are generally located within one node thus there will be fewer duplicates across nodes. This leads to more single occurrences of elements in the final sample that are then scaled by the DV-estimator, often resulting in estimates that are too high.

Thus, performance improves as clustering reduces since duplicates are increasingly scattered throughout the network. This results in higher quality samples as the sampler picks up an increasingly random set of values. These counts are scaled more accurately to produce better estimates.

4.2.3. 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 since it is easier to get accurate samples from such uniform distributions.

Thus, for highly clustered and/or skewed distributions, it is better to choose lower sub-sampling ratios at the nodes so that a higher quality sample is obtained by accessing more nodes. This however comes with the cost of longer random walks. Thus it is important for designers to keep this tradeoff in mind.

4.3. Supporting Different Queries

Because our algorithm is capable of having different parameters set for each random walk, each query can be handled differently. This offers great flexibility for supporting different queries.

When there is a set of known commonly executed queries and the P2P network is not highly dynamic, developers can study the properties of the data distribution on the P2P network by generating a set of graphs that show the algorithm's estimation quality (as seen in section 6.4). As seen in our experiments section, there is low variance across runs of the random walk and the results are fairly consistent. By investigating the performance of estimations, developers can choose appropriate values for the input parameters of our algorithm, such as the burn-in period and node and data sampling ratios. Alternatively, to support more complex and diverse queries, these parameters can be defined explicitly for each walk by administrators or power users, depending on the accuracy they require and the time constraints for running the query.

CHAPTER 5

EXPERIMENTATION

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

5.1. Implementation

Our algorithm was implemented in Java and carried out on Dual 3.00GHz Intel Xeon processors with 2GB RAM using the JUNG framework for graph generation [24]. The Java maximum heap size was set to 300MB.

5.1.1 Generation of P2P Networks

The P2P networks for running the algorithm were simulated by loading actual topologies of the Gnutella P2P network [17]. These were obtained from real-world data collected by M. Ripeanu [33] that was also used by Gunopulos et al. for their simulations in [2].

Our simulator loaded a total of 16.1 million integers onto 24278 peers by a total of 62391 edges. Each node was allotted a data set size between 300 and 1500 that indicates the size of its local database This distribution is considered to be Zipfian [14, 38] and several input parameters (described below) 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 a greater number of nodes with smaller-sized databases.

5.1.2 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. The multi-set is partitioned and the correct number of tuples is allocated during a breadth first traversal of the network. This is done to give control of clustering properties of the distribution. We can simulate different levels of clustering by shuffling the data in the multi-set before allocation. This changes the correlation of data between neighboring nodes. A highly shuffled multi-set has a lower level of clustering. This approach is adapted from [2].

On a P2P network like Gnutella, which is used for sharing songs and data files, some records tend to be more popular compared to others. We generate data for the nodes by following the intuition that the distribution of values follows Zipf's Law [14, 30, 38].

5.2. Input Parameters

A variety of input parameters allow different cases to be simulated to test our algorithm and its performance.

5.2.1 P2P Network Parameters

These parameters control the node sizes as they are allocated to all nodes in the graph.

- **Node Data Size Skew (z_{size}):** Controls the shape 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:** The highest and lowest possible number of tuples 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. 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 allowed node sizes.

5.2.2 Node Database Generation Parameters

These parameters control the properties of the distribution of tuples (integers) on the P2P network.

- **Number of Distinct Values:** The number of distinct data values on the entire P2P network.
- **Data Distribution Skew (z_{data}):** The data on the network follows a Zipfian frequency distribution. The skew controls the shape of this distribution. Ranging

from 0 to 1, lower values correspond to increasingly flat (uniform) distributions. Larger values have distributions that slant more steeply.

- **Cluster Level (Cl):** The level of mixing of the sorted data before it is allocated to the nodes. It ranges from 0 to 1 where 0 performs no mixing, and this corresponds to a perfectly clustered network, where each node and its neighbors hold very similar data. As the level tends to 1, data is mixed entirely before allocating to nodes.

5.2.3 Algorithm Input Parameters

The following parameters control the algorithm.

- **Burn-In Period (B):** The Metropolis-Hastings 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
- **Node Sub-Sampling Ratio (r_{node}):** The sampling ratio for the local repository of a node.
- **Data Sampling Ratio (r_{data}):** 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.

5.3. Evaluation Metrics

Evaluation of the algorithms is done by assessing the ratio error of the estimates [9]. It is given by:

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

The ideal estimator has a ratio error of 1. Other evaluation metrics could include the number of nodes sampled, the total size of samples collected or the number of messages exchanged.

5.4. Experiments and Results

We compare the results of a running a Metropolis-Hastings random walker with burn-in period of 10 with the results from a random sample of nodes in the system. This helps assess the quality of our traversal method. Because the variance across runs was low, each result is taken from just ten runs of the algorithm.

We also compare results with a uniform random sample of the data, taken directly from 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.

5.4.1 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 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 500,000 and we use AE to perform estimations at different sampling ratios.

At lower node sub-sample ratios, the error of the walk and the random node sample methods is close to that of the random data sample from the multi-set (for a fixed r_{data}). As the sub-sampling ratio increases, the sample collected from each node becomes more vulnerable to clustering and other data distribution factors. At higher sub-sampling ratios, more is sampled from a node and fewer nodes are sampled overall, thus the final sample is of lower quality with higher error.

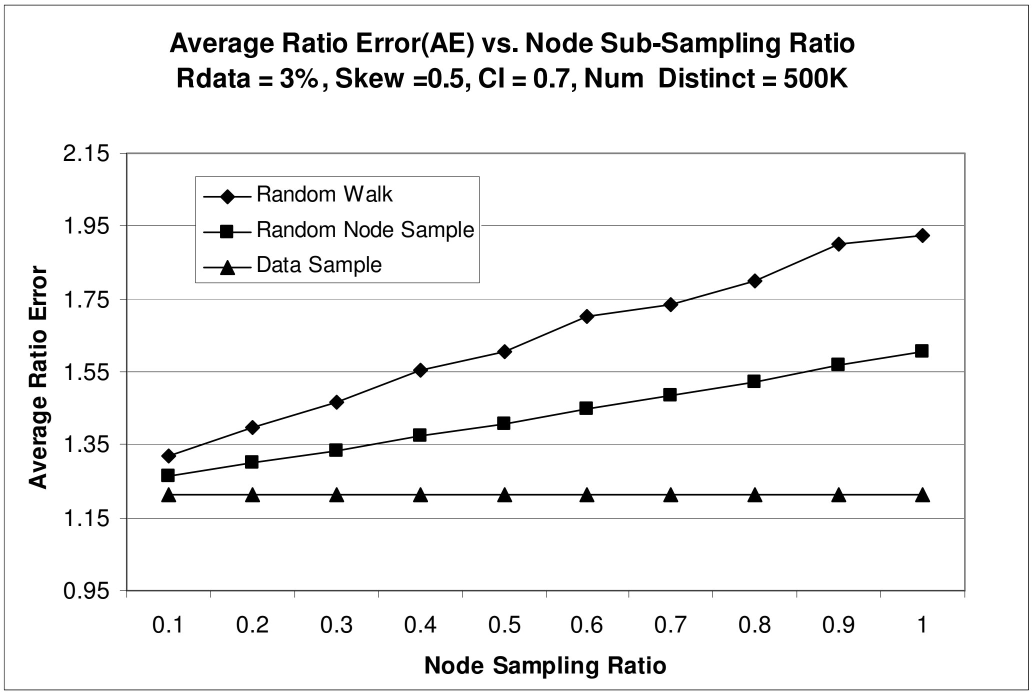

Figure 5: Average Ratio Error vs. Node Sub-Sampling Ratio

5.4.2 Data Sampling 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 node sub-sampling ratio (Rnode) is 0.5. It can also be seen that by setting the data sampling ratio to just 3%, the ratio error is close to 1 and comparable to a data sample. This gives insight into what ratio can be set when designing a walk for a particular application.

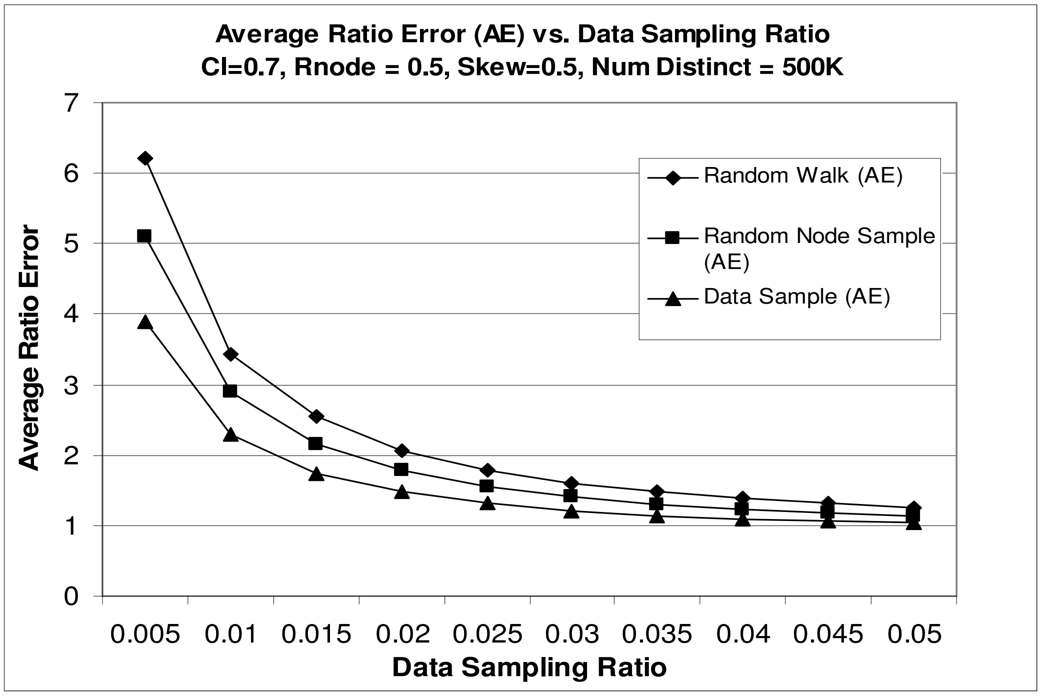

Figure 6: Average Ratio Error vs. Data Sampling Ratio

5.4.3 Clustering Level

Figure 7 shows the effect of varying the clustering level. As expected there is a high ratio error for perfectly clustered data. Estimation quality improves significantly when data is less clustered since a better, more varied sample results at the query initiator. This illustrates the importance of clustering in our sampling scheme, and that our algorithm actually leverages the extent of mixing on the network.

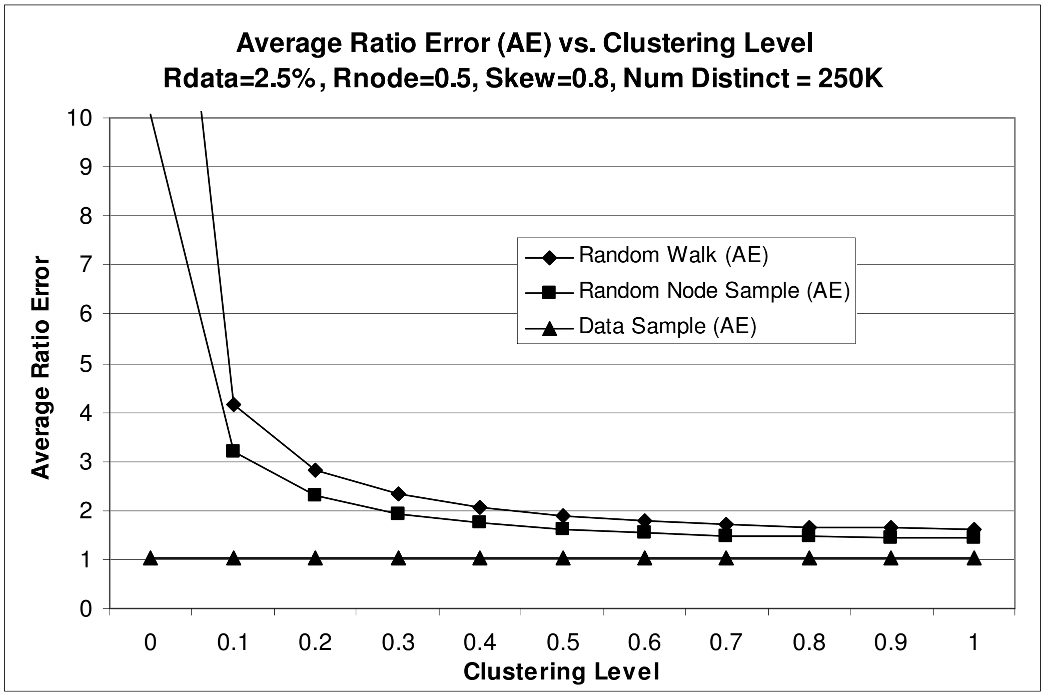

Figure 7: Average Ratio Error vs. Clustering Level

5.4.4 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 of AE increases greatly with a constant node sub-sampling ratio of 0.5. This is intuitive, because highly skewed data results in the algorithm picking up more duplicates of popular elements while sampling, and therefore fewer single occurrences of rarer elements. Since the distinct value estimator scales only single occurrences, the final estimate has a higher chance of error.

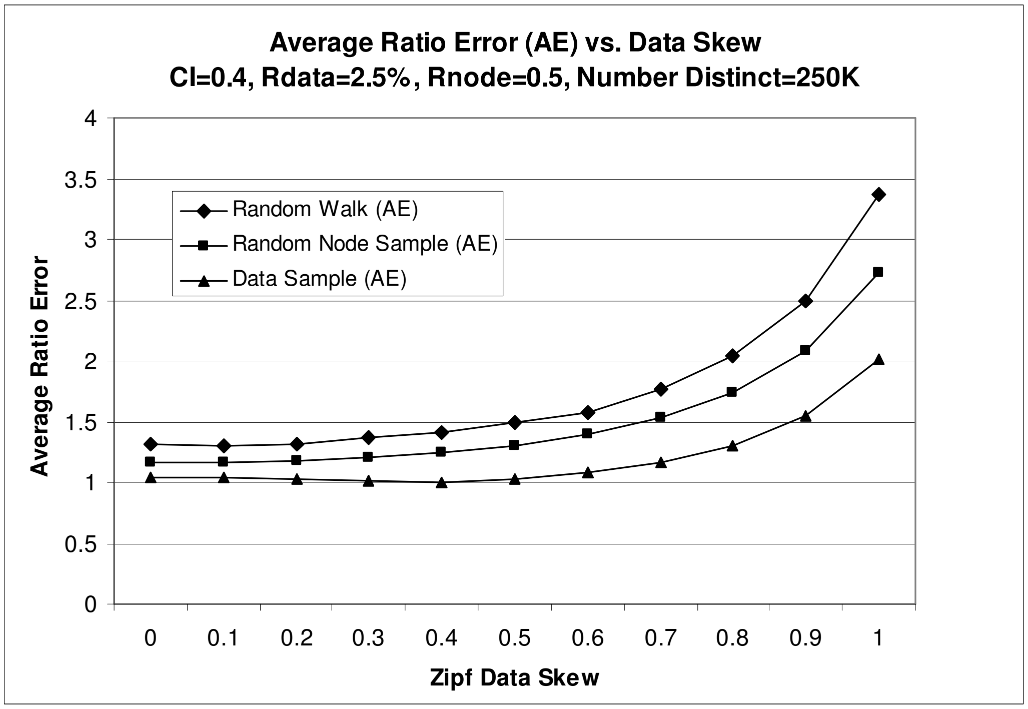

Figure 8: Average Ratio Error vs. Zipf Data Skew

5.4.5 Number of Distinct Values

We provide two graphs to demonstrate the operation of both GEE 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 by using GEE 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 it is a biased, optimal error estimator.

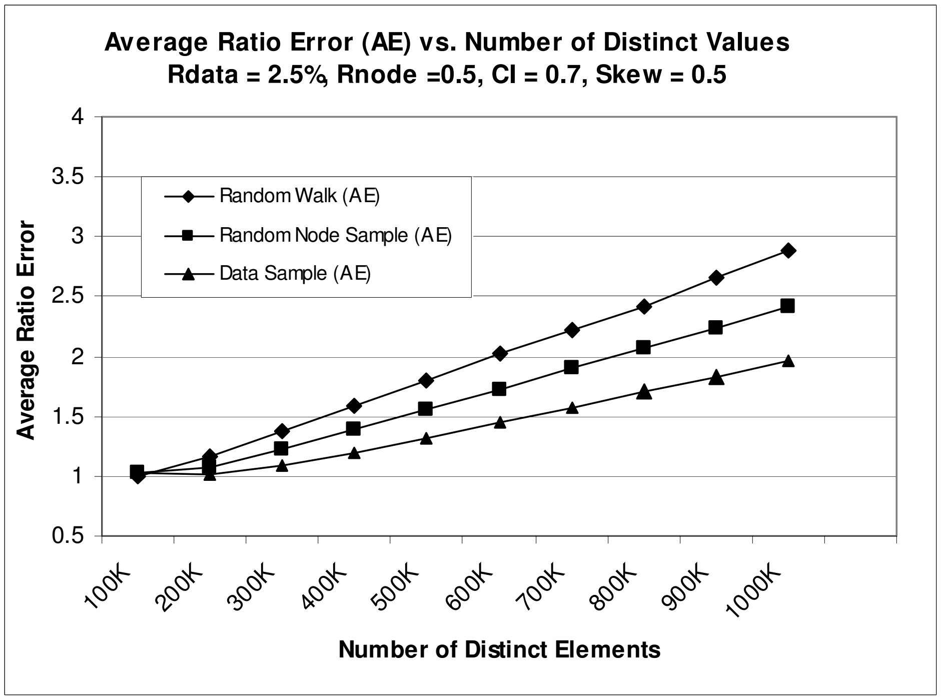

Figure 9: Average Ratio Error vs. Number of Distinct Elements using the Adaptive Estimator

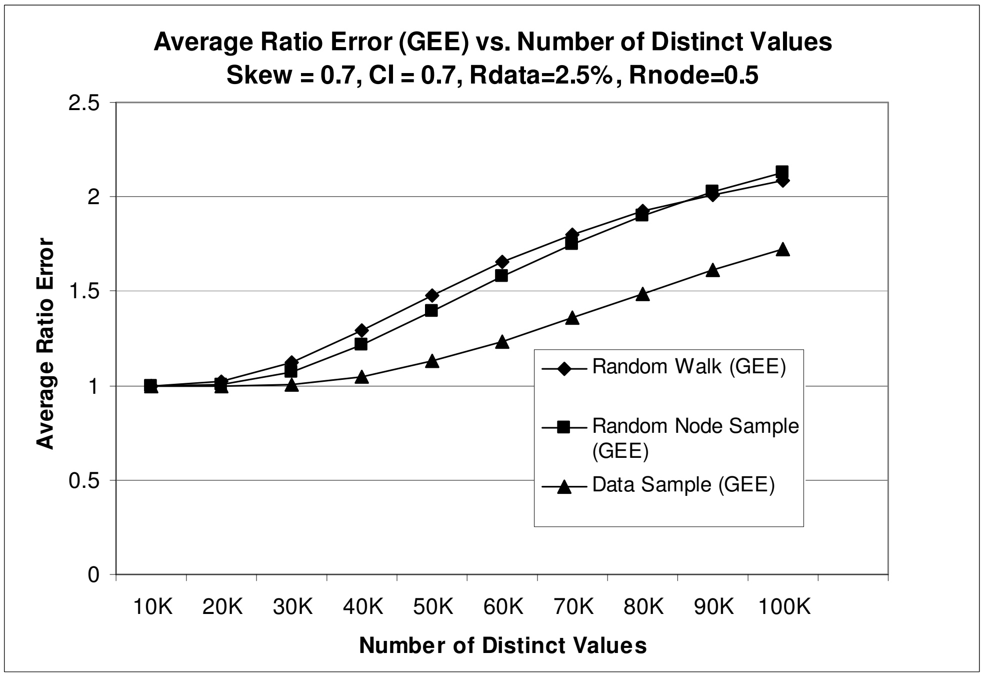

Figure 10: Average Ratio Error vs. Number of Distinct Elements using the Guaranteed Error Estimator

5.4.6 Burn-In Period

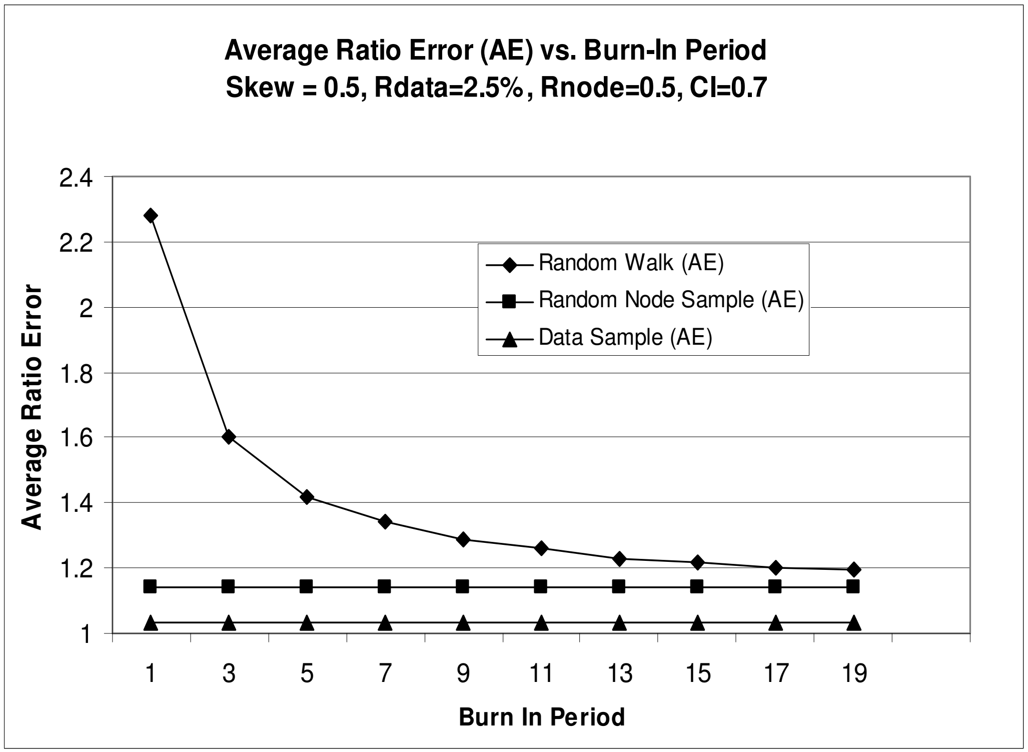

Figure 11: Average Ratio Error vs. Burn-In Period

Figure 11 shows the effect of executing the random walk with different burn-in periods. 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 data at sampled nodes becomes less correlated as the burn-in period increases.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis we have addressed 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, comparable to uniform-data samples. It also opens up numerous possibilities for future work.

An interesting area of further work is exploring how one can set node sub-sampling ratios so as to balance the quality of an estimate and its cost. It may even be possible to consider dynamic sub-sampling ratios, with different traversal techniques in order to get uniform-random samples from the network. Similarly, ways of predicting how much to sample from the entire P2P network warrant further study. Applying techniques of optimizing the sub-sampling process at each node can also be considered, such as maintaining a collection of samples at each node. This would reduce the cost of repeatedly sampling the local repositories at each node.

Methods of compressing the size of the samples sent back to the query initiator could also be investigated to cut down on network data transfer requirements. Being able to model the network requirements of accessing and retrieving samples from nodes also gives insight into setting input parameters for the algorithm, and also offer a means of predicting how best to bound a walk.

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

Being able to get distinct values estimations paves the way for future work in histogram construction, query optimization and duplicate elimination 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 computing models.

REFERENCES

[1] Androutsellis-Theotokis, S., and Spinellis, D. A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys, 36(4):335-371, Dec. 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] Awan, A., Ferreira, R.A., Jagannathan, S., and Grama, A. Distributed Uniform Sampling in Unstructured Peer-to-Peer Networks. In HICSS, 2006.

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

[5] Bar-Yossef, Z., and Gurevich, M. Random Sampling from a Search Engine's Index. International World Wide Web Conference Committee 2006

[6] Bar-Yossef, Z., Berg, A., Chien, S., Fakcharoenphol, J., and Weitz, D. Approximating Aggregate Queries about Web Pages via Random Walks. VLDB 2000.

[7] 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.

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

[9] Charikar, M., Chaudhuri, S., Motwani, R., and Narasayya, V. Towards Estimation Error Guarantees for Distinct Values. In Proceedings of the ACM PODS 2000.

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

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

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

[13] Flajolet, P., and Martin, G. N. Probabilistic counting algorithms for data base applications. J. Computer and System Sciences, 31:182–209, 1985.

[14] Ganesan, P., Bawa, M., and Garcia-Molina, H. Online balancing of range-partitioned data with applications to peer-to-peer systems .In VLDB 2004 (Toronto, Canada)

[15] Gibbons, P., Distinct Sampling for Highly-accurate Answers to Distinct Values Queries and Event Reports, VLDB 2001

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

[17] Gnutella Website: rfc-gnutella.sourceforge.net

[18] Goodman, L., On the estimation of the number of classes in a population. Annals of Math. Stat., 1949.

[19] Haas, P., and Swami, A. Sequential sampling procedures for query size estimation. 1992 SIGMOD

[20] Haas, P., Naughton, J., Seshadri, P., and Stokes, L. Sampling-based estimation of the number of distinct values of an attribute. In Proc. Of VLDB, 1995.

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

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

[23] Hou, W., Ozsoyoglu, G., and Taneja, B. Statistical estimators for relational algebra expressions. In Proc. of 1988 ACM PODS, pages 276–287, March 1988.

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

[25] KazaA website: kazaa.com

[26] Kempe, D., Dobra, A., and Gehrke, J. Gossip-Based Computation of Aggregate Information. In Proceedings. of the IEEE Symposium on Foundations of Computer Science, 2003

[27] King, V., and Saia, J. Choosing a random peer. In ACM Symp. On Principles of Distributed Computing, 2004

[28] Le Fessant, F., Handurukande, S., Kermarrec, A.-M., and Massoulié, L. Clustering in Peer-to-Peer File Sharing Workloads. IPTPS 2004

[29] Lipton, R., Naughton, J., and Schneider D. Practical selectivity estimation through adaptive sampling. 1990 ACM SIGMOD, pages 1–11, 1990.

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

[31] Metropolis, N., Rosenbluth A,, Rosenbluth, M., Teller, A., and Teller, E. Equations of state calculations by fast computing machines. J. of Chemical Physics, 1953.

[32] Ntarmos, N., Triantafillou, P., and Weikum, G. Counting at Large: Efficient Cardinality Estimation in Internet-Scale Data Networks, ICDE 2006.

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

[34] Shlosser, A. On estimation of the size of the dictionary of a long text on the basis of a sample. Eng. Cybernetics, 1981.

[35] Stutzbach, D., Rejaie, R., Duffield N., Sen, S., and Willinger, W. On Unbiased Sampling for Unstructured Peer-to-Peer Networks. IMC 2006

[36] Tsoumakos, D., and Roussopoulos, N. A Comparison of Peer-to-Peer Search Methods In Proc. of the 6th International Workshop on Web and Databases, 2003

[37] Zhong, M., and Shen, K. Random walk based node sampling in self-organizing networks. SIGOPS 2006

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

BIOGRAPHICAL INFORMATION

Zubin Joseph was born in Lusaka, Zambia and completed his Bachelor of Engineering in Lesotho, Southern Africa. He then took his Master’s degree at the University of Texas at Arlington, where his main research interests lie in database exploration and information retrieval applied in distributed environments.