P-2156

EFFICIENT APPROXIMATE QUERY PROCESSING IN PEER TO PEER NETWORKS

A PROJECT REPORT

Submitted by

ELANGO.V

RAVICHANDRAN.A

In partial fulfillment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641006

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2008

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"EFFICIENT APPROXIMATE QUERY PROCESSING IN PEER TO PEER NETWORKS"** is the bonafide work of,

ELANGO.V : 71204205006

RAVICHANDRAN.A : 71204205036

Who carried out the project under my supervision.

SIGNATURE	**SIGNATURE**
Dr.S.Thangasamy	Ms. J. Cynthia M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Department of Computer Science and Engineering, Kumaraguru College Of Technology, Coimbatore-641006.	Department of Information Technology, Kumaraguru College Of Technology, Coimbatore-641006.

Submitted for the university project viva voice held on

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our sincere thanks to our chairman **Padmabhushan Arutselvar Dr. N. Mahalingam B.Sc, F.I.E**, Vice Chairman **Dr. K. Arumugam B.E., M.S., M.I.E.,** Correspondent **Shri.M.Balasubramaniam** and **Joint Correspondent Dr.A Selvakumar** for all their support and ray of strengthening hope extended. We are immensely grateful to our principal **Dr. Joseph V. Thanikal M.E., Ph.D., PDF., CEPIT.,** for his invaluable support to the outcome of this project.

We are deeply obliged to **Dr.S.Thangasamy**, Dean, Department of Computer Science and Engineering for his valuable guidance and useful suggestions during course of this project.

We also extend our heartfelt thanks to our project coordinator **Prof.K.R.Baskaran B.E., M.S.**, Asst. Prof., Department of Information Technology for providing us his support which really helped us.

We are indebted to our project guide **Ms. J.Cynthia M.E.,** Sr.Lecturer, Department of Information Technology for her helpful guidance and valuable support given to us throughout this project.

We thank the teaching and non-teaching staffs of our Department for providing us the technical support during the course of this project. We also thank all of our friends who helped us to complete this project successfully.

DECLARATION

We,

Elango.V	Reg.No:	71204205006
Ravichandran.A	Reg.No:	71204205036

hereby declare that the project entitled **"Efficient Approximate Query Processing In Peer-to-Peer Networks"**, submitted in partial fulfillment to Anna University as the project work of Bachelor of Technology (Information Technology) degree, is a record of original work done by us under the supervision and guidance of the Department of Information Technology, Kumaraguru College of Technology, Coimbatore.

Place: Coimbatore

Date: 22.04.2008

[Elango.v] [Ravichandran.A]

Project Guided by

[Ms. J.Cynthia M.E.]

TABLE OF CONTENTS

ABSTRACT

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

LIST OF FIGURES

LIST OF SYMBOLS

P2P	**Peer to Peer**
AQP	**Approximate query processing**
COL	**Column**
CV ERROR	**Cross validation error**

CHAPTER 1

INTRODUCTION

1.1 Peer-to-Peer (P2P) Databases:

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

1.2 Aggregation Queries:

In this Project, however, we consider a problem on P2P systems that is different from the typical search and retrieval applications. As P2P systems mature beyond file sharing applications and start getting deployed in increasingly sophisticated e-business and scientific environments, the vast amount of data within P2P databases poses a

is, how aggregation queries on such databases can be answered. Aggregation queries have the potential of finding applications in decision support, data analysis, and data mining. For example, millions of peers across the world may be cooperating on a grand experiment in astronomy, and astronomers may be interested in asking decision support queries that require the aggregation of vast amounts of data covering thousands of peers. In addition, there is real-world value for aggregation queries in network monitoring scenarios such as temperature and anomaly detection in sensor networks, Intrusion Detection Systems and application signature analysis in P2P networks. Sensor networks can directly benefit from aggregation of traffic analysis data by offering a more efficient means of computing various network-based aggregates such as the average message size and maximum data throughput within the network, with minimal energy consumption and decreased response times. We make the problem more precise as follows: Consider a single table T that is distributed over a P2P system; that is, the peers store horizontal partitions (of varying sizes) of this table. An aggregation query such as the following may be introduced at any peer (this peer is henceforth called the query node). Aggregation query

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

In the above query, the Agg-Op may be any aggregation operator such as SUM, COUNT, AVG, and so on, Col may be any numeric measure column of T or even an expression involving multiple columns, and the selection condition decides which tuples should be involved in the aggregation. Although aggregation queries have been heavily [illegible] in traditional databases, it is not clear that these

techniques will easily adapt to the P2P domain. For example, decision support techniques such as online analytical processing (OLAP) commonly employ materialized views; however, the distribution and management of such views appear difficult in such a dynamic and decentralized domain. In contrast, the alternative of answering aggregation queries at runtime "from scratch" by crawling and scanning the entire P2P repository is prohibitively slow.

1.3 Approximate Query Processing (AQP):

Fortunately, it has been observed that in most typical data analysis and data mining applications, timeliness and interactivity are more important considerations than accuracy; thus, data analysts are often willing to overlook small inaccuracies in the answer, provided that the answer can be obtained fast enough. This observation has been the primary driving force behind the recent development of AQP techniques for aggregation queries in traditional databases and decision support systems. Numerous AQP techniques have been developed: The most popular ones are based on random sampling,

Where a small random sample of the rows of the database is drawn, the query is executed on this small sample, and the results are extrapolated to the whole database. In addition to simplicity of implementation, random sampling has the compelling advantage that, in addition to an estimate of the aggregate, one can also provide confidence intervals of the error, with high probability. Broadly, two types of sampling-based approaches have been investigated:

1. Precomputed samples, where a random sample is precomputed by scanning the database and the same sample is reused for several queries and

2. Online samples, where the sample is drawn "on the fly" upon encountering a query.

1.4 Goal of the Project:

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

Goal of the Project: Approximating Aggregation Queries in P2P Networks. Given an aggregation query and a desired error bound at a query node peer, compute with "minimum cost" an approximate answer to this query that satisfied the error bound. The cost of query execution in traditional databases is usually a straightforward concept: It is either I/O cost or CPU cost or a combination of the two. In fact, most AQP approaches simplify this concept even further by just trying to minimize the number of tuples in the sample, thus making the assumption that the sample size is directly related to the cost of query execution. However, in P2P networks, the cost of query execution is a combination of several quantities such as the number of participating peers, the bandwidth consumed (that is, the amount of data shipped over the network), the number of messages exchanged, the latency (the time to propagate the query across multiple peers and receive replies), the I/O cost of accessing data from participating peers, the CPU cost of processing data at participating peers, and so on. In this Project, we shall be concerned with information from the visited peers, such as the number of tuples, the aggregate of tuples (for example, SUM, COUNT, AVG, and so forth) that satisfy the selection condition, and send this information back to the query node. This information is then analyzed at the query node to determine the skewed nature of the data that is distributed across the network, such as the variance of the aggregates of the data at peers, the amount of correlation between tuples that exists within the same peers, the variance in the degrees of individual nodes in the P2P graph (recall that the degree has a bearing on the probability that a node will be sampled by the random walk), and so on. Once this data has been analyzed at the query node, an estimation is made on how much more samples are required (and in what way should these samples be collected) so that the original query can be optimally answered within the desired accuracy, with high probability.

replies) as our primary quantity to minimize though our technique could be easily extended to deal with other cost metrics.

1.5 Our Approach:

We briefly describe the framework of our approach. Essentially, we abandon trying to pick true uniform random samples of the tuples, as such samples are likely to be extremely impractical to obtain. Instead, we consider an approach where we are willing to work with skewed samples, provided that we can accurately estimate the skew during the sampling process. To get the accuracy in the query answer desired by the user, our skewed samples can be larger than the size of a corresponding uniform random sample that delivers the same accuracy; however, our samples are much more cost efficient to generate. Although we do not advocate any significant preprocessing, we assume that certain aspects of the P2P graph are known to all peers, such as the average degree of the nodes, a good estimate of the number of peers in the system, certain topological characteristics of the graph structure, and so on. Estimating these parameters via preprocessing are interesting problems in their own right; however, we omit these details from this Project. The main point that we make is that these parameters are relatively slow to change and thus do not have to be estimated at query time: It is the data contents of peers that changes more rapidly; hence, the random sampling process that picks a representative sample of tuples has to be done at runtime. Our approach has two major phases. In the first phase, we initiate a fixed-length random walk from the query node. This random walk should be long enough to ensure that the visited peers represent a close sample from the underlying stationary distribution (the appropriate length of such a walk

CHAPTER 2

RELATED WORK

P2P systems are becoming very popular because they provide an efficient mechanism for building large scalable systems [28]. Most recent work has focused on Distributed Hash Tables (DHTs) [32], [33], [37]. Such techniques provide scalability advantages over unstructured systems (such as Gnutella); however, they are not flexible enough for some applications, especially when nodes join or leave the network frequently or change their connections often. Recent work has proposed different techniques for exact query processing in P2P systems. Most proposals use structured overlay networks (DHTs), such as CAN, Pastry, and Chord. Such techniques include PIER, DIM, or Pastry, and since they use DHTs, they have a different focus and are not directly applicable to our case. A hybrid system, Mercury [4], using routing hubs to answer range queries was also recently proposed. This system is also designed to provide exact answers to range queries. These techniques use Markov-chain random walks to select random peers from the network. Their results show that when certain structural properties of the graph are known or can be estimated (such as the second eigenvalue of the graph), the parameters of the walk can be set so that a representative sample of the stationary distribution can be collected with high probability. There are known techniques for computing approximate aggregates in distributed settings (most notably, the Gossip protocol). The technique works generally as a preprocessing step where all peers in a network attempt to mix data among adjacent peers, eventually converging upon a single value. The inability to contact all nodes in the network makes it exceedingly difficult

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

CHAPTER 3

FOUNDATIONS OF OUR APPROACH

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

3.1 The Peer-to-Peer Model

We assume an unstructured P2P network represented as a graph G=(P,E),with a vertex set P ={P1,P2...Pm} and an edge set E. The vertices in P represent the peers in the network, and the edges in E represent the connections between the vertices in P. Each peer p is identified by the processor's IP address and a port number (IPp and portp). The peer p is also characterized by the capabilities of the processor on which it is located, including its CPU speed pcpu, memory bandwidth pmem, and disk space pdisk. The node also has a limited amount of bandwidth to the network, noted by pband. In unstructured P2P networks, a node becomes a member of the network by establishing a connection with at least one peer currently in the network. Each node maintains a small number of connections with its peers: The number of connections is typically limited by the resources at the peer. We denote the number of connections that a peer is maintaining by pconn. The peers in the network use the Gnutella P2P protocol to communicate. The Gnutella P2P protocol supports four message types (Ping, Pong, Query, and Query Hit), of which the Ping and Pong messages are used to establish connections with other peers, and the Query and Query Hit messages are used to search in the P2P network. Gnutella, however, uses a naive Breadth First Search (BFS) technique in which queries are propagated to all the peers in the network and thus consumes excessive network and processing resources and results in poor performance. Our approach, on the other hand, uses a probabilistic search algorithm based on random walks. The key idea is that technique is shown to improve the search efficiency and reduce unnecessary traffic in the P2P network.

3.2 Query Cost Measures

As mentioned in Section 1, the cost of the execution of a query in P2P databases is more complicated than equivalent cost measures in traditional databases. The primary cost measure that we consider is latency, which is the time that it takes to propagate the query across multiple peers and receive replies at the query node. In our algorithm, latency can be approximated by the number of peers that participate in the random walk. This measure is appropriate for our algorithm because it performs a single random walk starting from the query node. Thus, latency becomes proportional to the total number of visited peers in the random walk. To see this, we note that the aggregation operator (as well as the selection filter and IP address of the query node) can be pushed to each visited peer. Once a peer is visited by the algorithm, the peer can be instructed to simply execute the original query on its local data and send only the aggregate and the degree of the node back to the query node, from which the query node can reconstruct the overall answer. Moreover, this information can be sent directly without necessitating any intermediate hops, as the visited peer knows the IP address of the query node from which the query originated. This is reasonable, considering that the IP address can be pushed to visited peers along with the aggregation operator and the P2P networks such as Kazaa run on top of a Thus, the bandwidth requirement of such an approach is uniformly very small for all visited peers: They are not required to send more voluminous raw data (for example, all or parts of the local database) back to the query node. In approximating latency by the number of peers participating in the random walk, we also make the implicit assumption that the overhead of visiting peers dominates the costs of local computations (such as execution of the original query on the local database). This is, of course, true if the local databases are fairly small. To ensure that the local computations remain small even if local databases are large, our approach in such cases is to execute the aggregation query only on a small fixed-sized random sample of the local data (that is, we sub sample from the peer), scale the result to the entire local database, and send the scaled aggregate back to the query node. This way, we ensure that the local computations are uniformly small across all visited peers. In contrast, suppose that instead of a fixed sized sample, we decided on sampling a fixed fraction of a visited peer's local database. The main problem with this approach is that it complicates the query cost model. Now, local processing costs cannot be ignored and, thus, latency cost of executing a query cannot be modeled as simply being proportional to the number of visited peers (or even the overall number of sampled tuples). The latency now becomes a complex (and, perhaps, system dependent) function of the cost of visiting peers and local query processing costs. The consequence of a complicated latency model is that it now becomes difficult to have a principled two-phase approach to solving the problem because the first phase now has the task of determining how many peers should be sampled in the second phase so that the target accuracy can be achieved with minimum latency. Moreover, even if the first phase can somehow

latency cost of the second phase is unpredictable. It depends on the type of peers we visit, as peers with large databases will increase latency, whereas peers with small databases will decrease latency. In summary, for SUM and COUNT aggregates, latency is shown to be proportional to the number of peers participating in the random walk. Thus, our goal is to minimize the number of peers that must be visited in order to arrive at an approximate answer with the desired accuracy.

3.3 Random Walk

In Graphs In seeking a random sample of the P2P database, we have to overcome the sub problem of how a random sample of the peers themselves can be collected. Unrepresentative samples of peers can quickly skew results, producing erroneous aggregation statistics. Sampling in a nonhierarchical decentralized P2P network presents several obstacles in obtaining near-uniform random samples. This is because no peer (including the query node) knows the IP addresses of all other peers in the network: They are only aware of their immediate neighbors. If this were not the case, then, clearly, the query node could locally generate a random subset of IP addresses from among all the IP addresses and visit the appropriate peers directly. We note that this problem is not encountered in traditional databases, as even if one has to resort to cluster (or block-level) sampling such as in [11] and [18], obtaining an efficient sample of the blocks themselves is straightforward. This problem has been recognized in other contexts (see [16] and the references therein), and interesting solutions based on Markov-chain random walks have been proposed. We briefly review such approaches here. A Markov-chain random walk is a procedure that is initiated at the query node, and for

from among its neighbors (and itself and, thus, self loops are allowed). It is well known that if this walk is carried out long enough, then the eventual probability of reaching any peer p will reach a stationary distribution. To make this more precise,

let P = {p1; p2; . . . ; pM} be the entire set of peers,

Let E be the entire set of edges, and let the degree of a peer p be deg(Þ). Then, the probability of any peer p in the stationary distribution is

prob(p)=deg(p)/2|E|

It is important to note that the above distribution is not uniform: The probability of each peer is proportional to its degree. Thus, even if we can efficiently achieve this distribution, we will have to compensate for the fact that the distribution is skewed as above if we have to use samples drawn from it for answering aggregation queries. The main issue that has concerned researchers has been the speed of convergence.1 Most results have pointed to certain broad connectivity properties that the graph should possess for this to happen. In particular, it has been shown that if the transition probabilities that govern the random walk on the P2P graph are modeled as anM *M matrix, then the second eigenvalue2 plays an important role in these convergence results. The second eigenvalue describes connectivity properties of graphs, in particular, whether the graph has a small cut size,3 which would adversely impact the length of the walk necessary to arrive at convergence. As the results in [16] show, if the P2P graph is well connected (that is, it has a small second eigenvalue, and a minimum degree of the graph is large), then the random

certain specific conditions of connectedness (for example, expander graphs that are common in P2P networks), convergence can be achieved in O(logM) steps. In our case, recall from Section 1 that we assume that we are allowed a certain amount of preprocessing to determine various properties of the P2P graph that will be useful at query time (under the assumption that the graph topology changes less rapidly compared to the data content at the peers). The speed of convergence of a random walk in this graph is determined in this preprocessing step, in addition to other useful properties such as the number of nodes M, the number of edges jEj, and so on.

1. We define speed of convergence as how many hops h are necessary
2. Before one gets close to the stationary distribution.
3. The second eigenvalue tells how well the peers within the network are
4. Connected, that is, expander versus clustered sets of peers.
5. Given a partition of peers in two sets A and B, any edge crossing from A to be B is crossing the cut. The cut size is sum of the edges crossing A and B. convergence, we essentially determine a jump parameter j that determines how many peers can be skipped between selections of peers for the sample. As the jump increases, the correlation between successive peers that are selected for the sample decreases rapidly.

3.4 Sampling Theorems

In this section, we shall develop the formal sampling theorems that drive our algorithm. We shall show how the tuples that are retrieved from the first phase of our algorithm can be utilized to recommend how the second phase should be executed, that is, the "query plan" for answering the query approximately so that a desired error is achieved. We focus here on the COUNT aggregate for the purpose of illustrating our main ideas (our formal results can be easily extended for the SUM and AVERAGE cases). Finally, to keep the discussion simple, we assume that all local databases at peers are small, that is, sub sampling is not required (our results can be extended for the sub sampling case and, in fact, our algorithm in Section 4 does not make this assumption). As discussed earlier, our algorithm has two phases. In the first phase, our algorithm will visit a predefined number of peers m by using a random walk such that the sample of visited peers will appear as if they have been drawn from the stationary distribution of the graph. The query will be executed locally at each visited peer, and the aggregates will be sent back to the query node, along with other information such as the degrees of the visited peers (from which information such as the peers' probabilities in the stationary distribution can be computed). The query node analyzes this information and then determines how many more peers need to be visited in the second phase. The theorems that we develop next provide the foundations on which the decisions in the first phase are made.

Recall that P={P1,P2..Pm} is the set of peers.

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

Let the aggregate for a peer p be. Y(p) =? u€p y(u).

Let y be the exact answer for the query, that is,

y = ? p€py(p).

The query also comes with a desired error threshold ? req.

The implication of this requirement is that if y' is the estimated count by our algorithm, then

|y-y'| < ? req

Now, consider a fixed-size sample of peers

S = {s1, s2 . . . sm} where each si is from P. This sample is picked by the random walk in the first phase. We can approximate this process as that of picking peers in m rounds, where in each round, a random peer si is picked from P, with probability prob(si) .We also assume that peers may be picked with replacement; that is, multiple copies of the same peer may be added to the sample, as this greatly simplifies the statistical derivations below. Consider the quantity y"defined as follows:

Y"= ? y(s)/prob(s)/m

s€S

To extend to any m, we make use of the following formulas for variance:

1. Var[aX]= a2Var[X] and
2. Var[X +Y] = Var[X_]+ Var[Y], where X and Y are independent random variables, and a is a constant. By using these formulas, we can easily show that Var[y"] C/m. The above Standard Error Theorem shows that the variance varies inversely as the sample size. The quantity C also represents the "badness" of the clustering of the data in the peers: The larger the C, the more the correlation among the tuples within peers and, consequently, the more peers need to be sampled to keep the variance of the estimator y" small. Notice also that if we divide the variance by N2, then we will effectively get the square of the error of the relative count aggregate if y" was used as an estimator for y. Our case is actually the reverse; that is, we are given a desired error threshold ? req, and the task is to determine the appropriate number of peers to sample that will satisfy this threshold. Of course, we have used a fixed-sized m in the first phase, so unless we are simply lucky, it is unlikely that this particular m will satisfy the desired accuracy. However, we can use the first phase more carefully to determine the appropriate sample size to draw in the second phase, say, m0. The main task is to use the sample drawn in the first phase to try to estimate C because once we estimate C, we can determine m0 by using Theorem 2.

Theorem 1.

E[y"]=y, that is, y", is an unbiased estimator of y.

Proof

Intuitively, each sampled peer s tries to estimate y as y(s)/prob(s), that is, by scaling its own aggregate by the inverse of its probability of getting picked. The final estimate y" is simply the average of the m individual estimates. To proceed with the proof, consider the simple case of only one sampled peer, that is, m ¼ 1. In this case, E[y"]=? (y(p)/prob(p))=y. To extend to any m, we make use of the linearity of expectation formula E[X+Y]=E[X]+E[Y] for random variables X and Y (that need not even be independent). Thus, if the expected estimate of any single random peer is y, then the expected average estimate by m random peers is also y. tu We next need to determine the variance of the random variable y".

Theorem 2 (Standard Error Theorem):

Var[y"]=? (y(p)/prob(p)-y)2 prob(p)/m
p€P

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

C=? (y(p)/prob(p)-y)2 prob(p).
p€P

3. We suggest a simple cross-validation procedure as described below to estimate C

Consider two random samples of peers of size m, each drawn from the stationary distribution. Let y1" and y2" be the two estimates of y by these samples, respectively, according to (1). We define the cross-validation error (CVError) as CV Error =[y1"-y2"].

Theorem 3:

E[CV Error2]=2E[(y"-y)2].

Proof.

E[CV Error2] =E [(y1"-y2")]=E[(y" – y)2] + E[(y2"-y)2] = 2E[(y"-y)2].

This theorem says that the expected value of the square of the CVError is two times the expected value of the square of the actual error. This CVError can be estimated in the first phase by the following procedure. Randomly divide the m samples into halves and compute the CVError (for sample size m=2). We can then determine C by fitting this computed error and the sample size m=2 into the equation in Theorem 2. To get a somewhat more robust estimation for C, we can repeat the random halving of the sample collected in the first phase several times and take the average value of C. We also note that since the CVError is larger than the true error, the value of C is conservatively overestimated. Once C is determined (that is, the "badness" of the clustering of data in the peers), we can determine the right number of peers to sample in the

CHAPTER 4

OUR ALGORITHM

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

4.1 COUNT

Our approach in the first phase is broken up into the following main components. First, we perform a random walk on the P2P network, attempting to avoid skewing due to graph clustering and vertices of high degree. Our walk skips j nodes between each selection to reduce the dependency between consecutive selected peers. As the jump size increases, our method increases overall bandwidth requirements within the database, but for most cases, small jump sizes suffice for obtaining random samples. Second, we compute aggregates of the data at the peers and send these back to the query node. Note that in Section 3, we had not formally discussed the issue of sub sampling at peers: This was primarily done to keep the previous discussion simple. In reality, the local databases at some peers can be quite large, and aggregating them in their entirety may not be negligible as compared to the overhead of visiting the peer. In other words, the simplistic cost model of only counting the number of visited peers is inappropriate. In such cases, it is preferable to randomly sub sample a small portion of the local database and apply the aggregation only to this sub sample. Thus, the ideal approach for this problem is to develop a cost model that takes into account cost of visiting peers, as well as local processing costs. Moreover, for such cost models, an ideal two-phase algorithm should determine various parameters in the first phase, such as how many peers should be visited in the second phase and how many tuples should be sub sampled from each visited peer. In this Project, we have taken a somewhat simpler approach in which we fix a constant t (determined at preprocessing time via experiments) such that if a peer has at most t tuples, then its database is aggregated in its entirety, whereas if the peer has more than t tuples, then t tuples are randomly selected and aggregated. Sub sampling can be more efficient than scanning the entire local database, for example, by block level sampling, in which only a small number of disk blocks are retrieved. If the data in the disk blocks are highly correlated, then it will simply mean that the number of peers to be visited will increase, as determined by our cross validation approach at query time. Third, we estimate the CVError of the collected sample and use that to estimate the additional number of peers that need to be visited in the second phase. For improving robustness, steps 2-4 in the cross-validation procedure can be repeated a few times, as well as the average squared CVError computed. Once the first phase has completed, the second phase is then straightforward. We simply initiate a second random walk based on the recommendations of the first phase and compute the final aggregate.

4.2 SUM and AVERAGE

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

4.3 MEDIAN

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

1. Select m peers at random by using random walk.
2. Each peer sj computes its median medj and sends it
3. To the query node, along with probðsjÞ.
4. The query node randomly partitions the m medians
5. Into two groups of m/2 medians: Group1 and Group2.
6. Let medg1 be the weighted median of Group1, that is,
7. Such that the following is minimized:

abs(? 1/prob(sj) - ? 1/prob(sj)).
medjЄGroup1 medjЄGroup1
medj<medg1 medj>medg1

Find the error between the median of Group2 (say, medg2) and the weighted rank of medg1 in Group2.

That is, let

c = abs (? 1/prob(sj) - ? 1/prob(sj)) /(m/2)
medjЄGroup2 medj<group2
medj<medg1 medj<medg2

6. Select additional c2/? 2 req peers by using random walk.

7. Find and return the weighted median of the medians of the additional peers.

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

CHAPTER 5

HYBRID SOLUTION

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

Algorithm: COUNT queries

Predefined values

M : total number of peers in network

E : total number of edges in network

m : number of peers to visit in phase 1

j : jump size for random walk

t : max #tuples to be subsampled per peer Inputs

Q : COUNT query with selection condition

Sink : peer where query is initiated

? req : desired max error

Phase 1

// Perform random walk

1. Curr = Sink; Hops =1;
2. while (Hops < j * m) {
3. if (Hops % j)
4. V isit(Curr);
5. Hops++;
6. Curr = random adjacent peer
7. }

// Visit peer

1. Visit(CurrÞ){
2. if (#tuples of Curr) <= t) {
3. Execute Q on all tuples
4. else
5. Execute Q on t randomly sampled
7. }
8. y(Curr) =(#tuples / #processed tuples)*(result_of_Q)
9. Return (y(Curr), deg(Curr)) to Sink
10. }

// Cross validate at sink

1. Let S ={s1; s2; . . . ; sm} be the visited peers
2. Partition S randomly into halves: S1 and S2
3. Compute

$$y1'' = \sum_{s \in s1} y(s)/prob(s)/(m/2)$$

$$y2'' = \sum_{s \in s2} y(s)/prob(s)/(m/2)$$

where prob(s)= deg(s)/2E.

4. Compute CV Error = |y1''-y2''|
5. Return m'=(m/2) * (CVERROR2/? 2req)

Phase 2

1. Visit m' peers by using random walk
2. Let S'={s1, s2. . . s'm}be the visited peers
3. 3. Return y'= ? y(s)/prob(s)/m'

CHAPTER 6

EXPERIMENTAL EVALUATION

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

6.1 Implementation:

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

6.2 Generation of P2P Networks and Databases:

6.2.1 Synthetic Topology:

The power laws offer insight to the structure of Internet topologies, and confirms that the power laws extend to P2P networks. Our synthetic topology is created through the process of connecting sub graphs by using the graph generation tool Jung. It consists of 10,000 peers and 100,000 edges. The parameters during graph creation are . Sub graphs (s). s sub graphs are created, which follow the power-laws topology. Edges between sub graphs (e). The size of edetermines the cut size between sub graphs. As the cut size decreases, the number of edges between sub graphs decreases.

6.2.2 Real-World Topology

It is also experimented with 2001 Gnutella topology data containing 22,556 peers and 52,321 edges, acquired from the data.

6.3 P2P Databases

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

Cluster Level (CL)

If the CL is equal to 0, then the data set is perfectly clustered; that is, it is sorted and then partitioned across the peers. If the CL is set to 1, then the data set is randomly permuted then partitioned across the peers. In-between values correspond to in-between scenarios.

Skew (Z)

The skew determines the slant in frequency distribution of distinct values in the data. Low skew values give the data set an even distribution of frequencies per value; conversely, high skew values distort the distribution of frequencies. We populated the synthetic network with 1,000,000 tuples and the Gnutella network with 2,200,000 tuples. It is well known that P2P databases have strong clustering properties; for example, large networks such as Gnutella contain subgraphs of peers containing similar music genre, movies, software, or documents. Thus, breadth first method in order to obtain reasonable clustering of synthetic data within the topologies. That is, when loading a peer, the adjacent peers are also loaded with similarly clustered data.

6.4 Input Parameters

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

1. Required Accuracy (? req). This parameter defines the maximum allowed error for the estimated answer.
2. Tuples Sampled per Peer (t). This parameter defines the number of tuples to be sampled from each selected peer.
3. Jump Size (j). This parameter defines the number of peers to be passed over before selecting the next peer for sampling.
4. Initial Sample Size (rorig). This parameter defines the initial number of tuples to be acquired from the database to execute the first phase. (Thus, rorig/t = m, where m is the number of peers visited in the first phase. In our experiments, the local databases are always large enough to ensure that sub sampling always takes place.) Parameter 1 is provided by the user for each query. Parameters 2-4 may be provided by the user or may be set via a preprocessing step.

6.5 Evaluation Metrics: Cost and Accuracy

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

6.6 Experiments

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

6.6.1 Accuracy

Figs. 1 and 2 show representative accuracy results for COUNT by using synthetic and real data sets. In this case, we have a query with selectivity of 30 percent, CL = 0.2, and Z = 0.2. In Fig. 2, we vary the required accuracy. The figure shows that the algorithm's result is always within the required accuracy. In Fig. 3, we set required accuracy to 0.1 and show the resulting accuracy for each query with different selectivity.

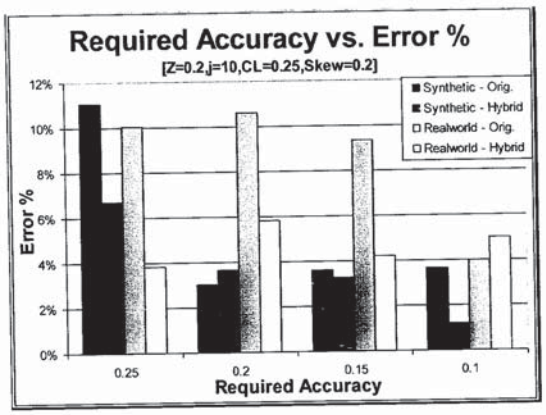


Figure 01.Required accuracy on the error percentage For the count technique

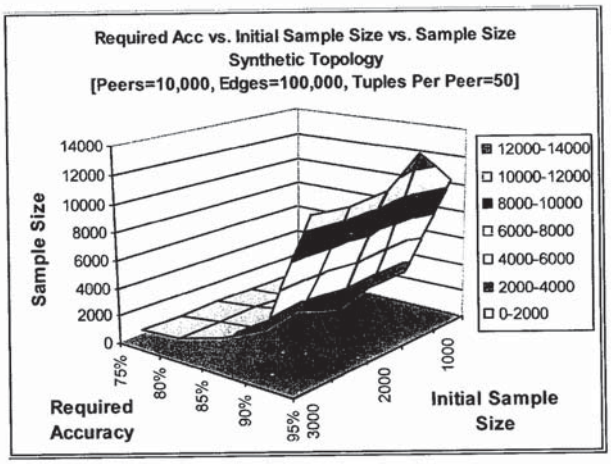
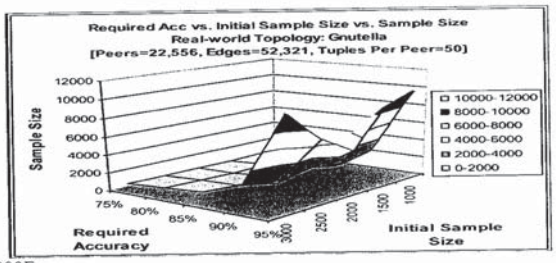


Figure 02. Effects of selectivity on the error percentage for the Count technique

6.6.2 Sample Size

Figs. 3 and 4 show that the required sample size increases with 1=? 2 req. They also show that the required sample size does not vary much when the initial sample is ranged from 1,000 to 3,000. The selectivity of the query in this experiment was 30 percent, and the



0000E

Figure 04. Effects of the sample size collected for given required accuracies and initial sample sizes for the count technique.

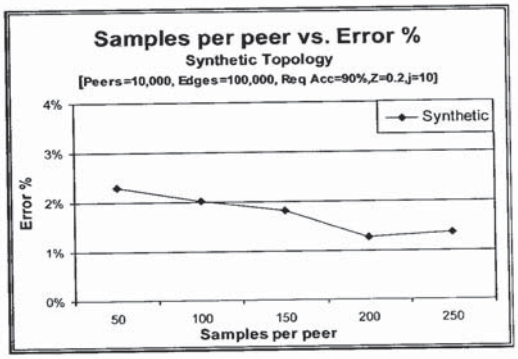


Figure 05. The number of peers does not make a vast difference in accuracy

note that the result of our algorithm specifies the number of peers to be sampled. In the experiments, we convert it to the number of samples by taking 25 samples per peer. Fig. 5 shows that the improvement by getting more tuples per peer is small. To minimize the cost of sampling in each peer, we take 25 samples in each peer.

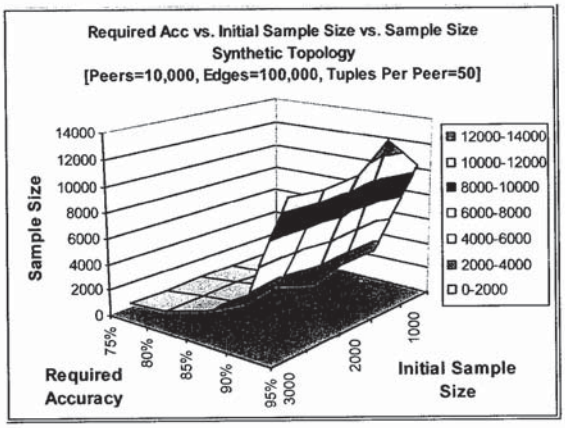


Figure 03.Effects of the sample size collected for given required accuracies and initial sample sizes for the Count technique.

6.6.3. Evaluating the SUM Query:

Figs. 06 and show that our technique provides similar accuracy results for SUM. Here, we estimate the SUM of all tuples in the database (that is, selectivity = 1). Fewer peers is highly dependent upon the clustering of the data. there is a direct relationship between clustering and the number of tuples sampled from each peer: Highly clustered networks can be better estimated using smaller sample sizes per peer; inversely, networks with little or no clustering can be estimated using larger samples sizes per peer, reducing the total number of messages sent over the network.

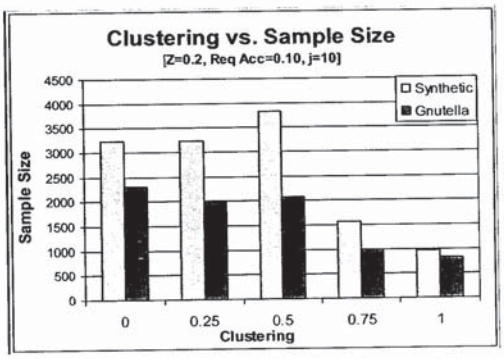


Figure 06. Effects of clustering on the sample size for the SUM technique.

6.6.4 Estimating the MEDIAN:

Figs. 07 illustrate that our technique can be extended to accurately estimate the MEDIAN. Similarly to SUM and COUNT aggregates our technique for computing the MEDIAN performs well by using various levels of clustering. In these experiments, we use both the Gnutella and the synthetic graphs, vary the clustering factor, and set ? req = 0.1. The error that we show in the graph is the difference between the true rank of the median that the algorithm returns and N=2

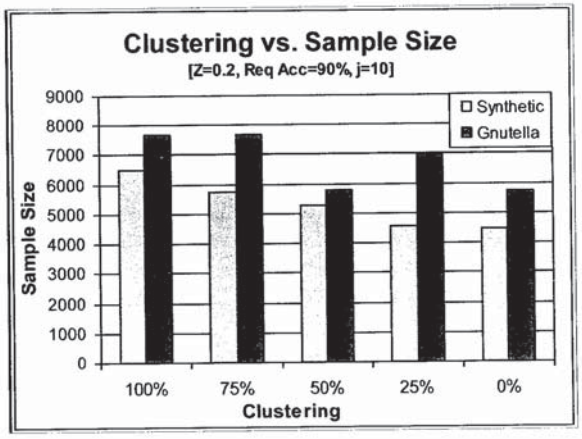


Figure 07. Effects of clustering on the sample size for the MEDIAN technique

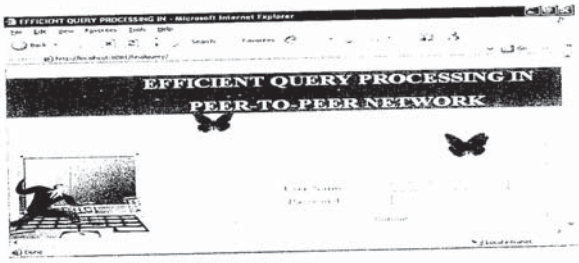
CHAPTER 7

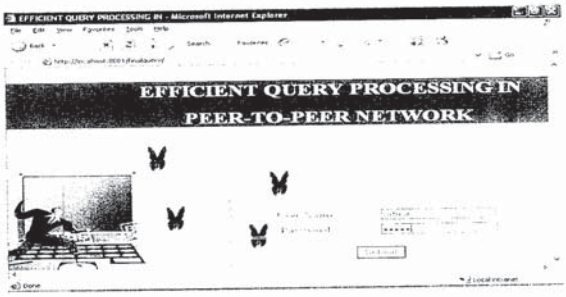
CONCLUSION

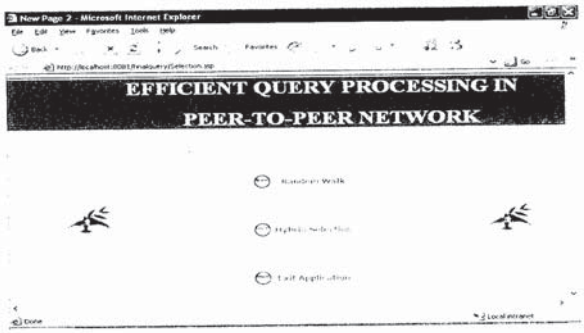
In this Project, we present adaptive sampling-based techniques for the approximate answering of ad hoc aggregation queries in P2P databases. Our approach requires a minimal number of messages sent over the network and provides tunable parameters to maximize performance for various network topologies. Our approach provides a powerful technique for approximating aggregates of various topologies and data clustering but comes with limitations based upon a given topologies structure and connectivity. For topologies with very distinct clusters of peers (small cut size), it becomes increasingly difficult to accurately obtain random samples due to the inability of random-walk process to quickly reach all clusters. This can be resolved by increasing the jump size, allowing a larger number of peers to be considered and increasing the allowed mixing by our hybrid approach. By varying a few parameters, our algorithm successfully computes aggregates within a given required accuracy. We present extensive experimental evaluations to demonstrate the feasibility of our solutions for both synthetic and real-world topologies.

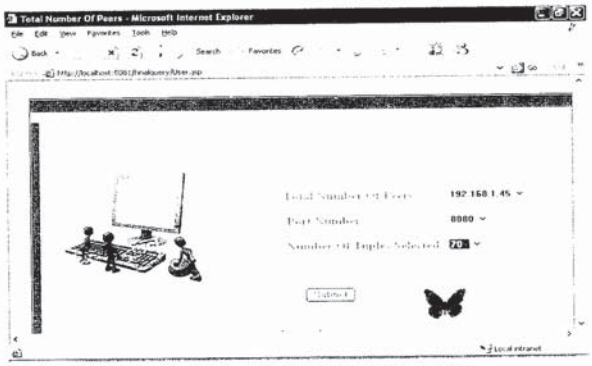
CHAPTER -8 APPENDICES

A. SCREEN LAYOUT

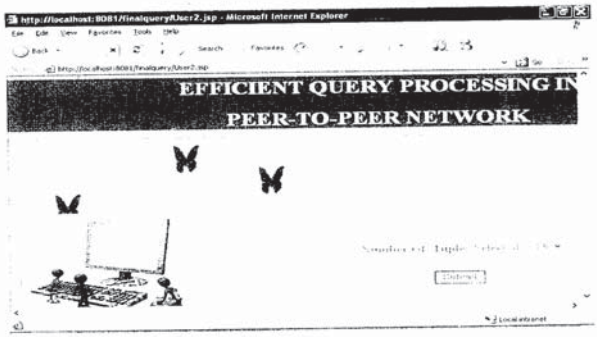


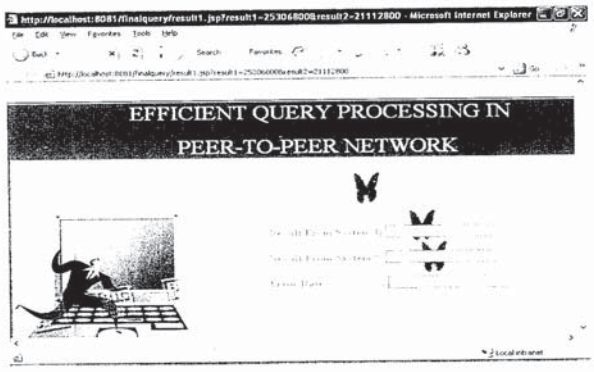



New Page 2 - Microsoft Internet Explorer
EFFICIENT QUERY PROCESSING IN
PEER-TO-PEER NETWORK


Total Number Of Peers - Microsoft Internet Explorer
192.168.1.45
8080


Tomcat


http://localhost:8081/finalqueryUser2.jsp - Microsoft Internet Explorer
EFFICIENT QUERY PROCESSING IN
PEER-TO-PEER NETWORK


http://localhost:8081/finalqueryresult1.jsp?result1=25306800&result2=21112800 - Microsoft Internet Explorer
EFFICIENT QUERY PROCESSING IN
PEER-TO-PEER NETWORK

B. CODINGS:

```
<%@ page language="java" %>        //header file(dir include)//
<%@ page import="java.sql.*" %>
<%!                                //global declaration//
String uname,pword;
%>

<%                                 //coding//

uname=request.getParameter("T1");
pword=request.getParameter("T2");
System.out.println(uname);
System.out.println(pword);

%>
<%
if(uname!=null)
{
if(uname.equals("admin"))
{
if(pword.equals("admin"))
{

response.sendRedirect("Selection.jsp");

}

}
}

%>

<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-
1252">
<title>EFFICIENT QUERY PROCESSING IN</title>
</head>
```

```
<body>

<div style="position: absolute; width: 901px; height: 527px; z-index: 1;
left: 47px; top: 33px; border-style: solid; border-width: 1px"
id="layer1">

        <script language="JavaScript1.2">

/*
Flying Butterfly script (By BGAudioDr@aol.com)
Modified slightly/ permission granted to Dynamic Drive to feature script
in archive
For full source, visit http://www.dynamicdrive.com
*/

var Ymax=8;                    //MAX # OF PIXEL STEPS IN THE
"X" DIRECTION
var Xmax=8;                    //MAX # OF PIXEL STEPS IN THE
"Y" DIRECTION
var Tmax=10000;                //MAX # OF MILLISECONDS
BETWEEN PARAMETER CHANGES

//FLOATING IMAGE URLS FOR EACH IMAGE. ADD OR DELETE
ENTRIES. KEEP ELEMENT NUMERICAL ORDER STARTING
WITH "0" !!

var floatimages=new Array();
floatimages[0]='butterfly2.gif';
floatimages[1]='butterfly2.gif';
floatimages[2]='butterfly2.gif';
floatimages[3]='butterfly2.gif';

//********DO NOT EDIT BELOW**********
var NS4 = (navigator.appName.indexOf("Netscape")>=0 &&
parseFloat(navigator.appVersion) >= 4 &&
parseFloat(navigator.appVersion) < 5)? true : false;
var IE4 = (document.all)? true : false;
var NS6 = (parseFloat(navigator.appVersion) >= 5 &&
navigator.appName.indexOf("Netscape")>=0 )? true: false;
var wind_w, wind_h, t='', IDs=new Array();
```

```
t+=(NS4)?'<layer name="pic'+i+'" visibility="hide" width="10"
height="10"><a href="javascript:hidebutterfly()">' : '<div id="pic'+i+'"
style="position:absolute; visibility:hidden;width:10px; height:10px"><a
href="javascript:hidebutterfly()">';
t+='<img src="'+floatimages[i]+'" name="p'+i+'" border="0">';
t+=(NS4)? '</a></layer>':'</a></div>';
}
document.write(t);

function moveimage(num){
if(getidleft(num)+IDs[num].W+IDs[num].Xstep >=
wind_w+getscrollx())IDs[num].Xdir=false;
if(getidleft(num)-IDs[num].Xstep<=getscrollx())IDs[num].Xdir=true;
if(getidtop(num)+IDs[num].H+IDs[num].Ystep >=
wind_h+getscrolly())IDs[num].Ydir=false;
if(getidtop(num)-IDs[num].Ystep<=getscrolly())IDs[num].Ydir=true;
moveidby(num, (IDs[num].Xdir)? IDs[num].Xstep :  -IDs[num].Xstep ,
(IDs[num].Ydir)?  IDs[num].Ystep:  -IDs[num].Ystep);
}

function getnewprops(num){
IDs[num].Ydir=Math.floor(Math.random()*2)>0;
IDs[num].Xdir=Math.floor(Math.random()*2)>0;
IDs[num].Ystep=Math.ceil(Math.random()*Ymax);
IDs[num].Xstep=Math.ceil(Math.random()*Xmax)
setTimeout('getnewprops('+num+')', Math.floor(Math.random()*Tmax));
}

function getscrollx(){
if(NS4 || NS6)return window.pageXOffset;
if(IE4)return document.body.scrollLeft;
}

function getscrolly(){
if(NS4 || NS6)return window.pageYOffset;
if(IE4)return document.body.scrollTop;
}

function getid(name){
if(NS4)return document.layers[name];
```

```
if(NS6)return document.getElementById(name);
}
function moveidto(num,x,y){
if(NS4)IDs[num].moveTo(x,y);
if(IE4 || NS6){
IDs[num].style.left=x+'px';
IDs[num].style.top=y+'px';
}}

function getidleft(num){
if(NS4)return IDs[num].left;
if(IE4 || NS6)return parseInt(IDs[num].style.left);
}

function getidtop(num){
if(NS4)return IDs[num].top;
if(IE4 || NS6)return parseInt(IDs[num].style.top);
}

function moveidby(num,dx,dy){
if(NS4)IDs[num].moveBy(dx, dy);
if(IE4 || NS6){
IDs[num].style.left=(getidleft(num)+dx)+'px';
IDs[num].style.top=(getidtop(num)+dy)+'px';
}}

function getwindowwidth(){
if(NS4 || NS6)return window.innerWidth;
if(IE4)return document.body.clientWidth;
}

function getwindowheight(){
if(NS4 || NS6)return window.innerHeight;
if(IE4)return document.body.clientHeight;
}

function init(){
wind_w=getwindowwidth();
wind_h=getwindowheight();
for(i=0; i<floatimages.length; i++){
```

```
if(NS4){
IDs[i].W=IDs[i].document.images["p"+i].width;
IDs[i].H=IDs[i].document.images["p"+i].height;
}
if(NS6 || IE4){
IDs[i].W=document.images["p"+i].width;
IDs[i].H=document.images["p"+i].height;
}
getnewprops(i);
moveidto(i , Math.floor(Math.random()*(wind_w-IDs[i].W)),
Math.floor(Math.random()*(wind_h-IDs[i].H)));
if(NS4)IDs[i].visibility = "show";
if(IE4 || NS6)IDs[i].style.visibility = "visible";
startfly=setInterval('moveimage('+i+')',Math.floor(Math.random()*100)+
100);
}}

function hidebutterfly(){
for(i=0; i<floatimages.length; i++){
if (IE4)
eval("document.all.pic"+i+".style.visibility='hidden'")
else if (NS6)
document.getElementById("pic"+i).style.visibility='hidden'
else if (NS4)
eval("document.pic"+i+".visibility='hide'")
clearInterval(startfly)
}
}

if (NS4||NS6||IE4){
window.onload=init;
window.onresize=function(){ wind_w=getwindowwidth();
wind_h=getwindowheight(); }
}

</script>

        <div style="position: absolute; width: 899px; height: 64px; z-
index: 1; left: 0px; top: 0px; background-color: #0099CC" id="layer2">
                <font
sp;         &nb
sp; 
                <font
color="#FFFFFF">       &
nbsp;    
                <b>EFFICIENT QUERY PROCESSING IN
</b></font></font>
                <p><b><font size="6"
color="#FFFFFF">       &
nbsp;         &
nbsp;         &
nbsp;         &
nbsp;    
                PEER-TO-PEER NETWORK</font></b></div>
        <div style="position: absolute; width: 897px; height: 29px; z-
index: 2; left: 0px; top: 495px; background-color: #0099CC"
id="layer3">
 </div>
        <div style="border:3px double #0099CC; position: absolute; width:
438px; height: 238px; z-index: 3; left: 398px; top: 141px" id="layer4">

                <div style="position: absolute; width: 28px; height: 428px;
z-index: 1; left: 601px; top: -8px; background-color: #0099CC"
id="layer7">
 </div>
                <form method="get" action="index.jsp">
                        <!--webbot bot="SaveResults" U-File="C:\Documents
and Settings\Antony\Desktop\peer\_private\form_results.csv" S-
Format="TEXT/CSV" S-Label-Fields="TRUE" -->
                        <p> </p>
                        <p> </p>
                        <p> </p>
                        <table border="1" width="100%" id="table1"
bordercolorlight="#FFFFFF" bordercolordark="#FFFFFF">
                                <tr>
                                        <td width="205">
                                        <p align="center"><b><font size="4"
color="#0099CC">User
                                        Name</font></b></td>
                                        <td>
                                        <input type="text" name="T1" size="20"
style="color: #0099CC; font-weight: bold; border-style: solid; border-
width: 2px"></td>
                                </tr>
                                <tr>
                                        <td width="205">
                                        <p align="center"><b><font
color="#0099CC" size="4">Password</font></b></td>
                                        <td>
                                        <input type="password" name="T2"
size="20" style="color: #0099CC; font-weight: bold; border-style: solid;
border-width: 2px"></td>
                                </tr>
                        </table>
                        <p align="center">
                        <input type="submit" value="Submit" name="B1"
style="color: #0099CC; font-weight: bold; border-style: ridge; border-
width: 2px"></p>
                        <p align="center"> </p>
                </form>
                <p> </div>
        <div style="position: absolute; width: 231px; height: 388px; z-
index: 4; left: 39px; top: 104px" id="layer5">
                <div style="position: absolute; width: 33px; height: 433px;
z-index: 1; left: -36px; top: -13px; background-color: #0099CC"
id="layer6">
 </div>
                <p> </p>
                <p> </p>
                <p>
                <img border="0" src="874256.jpg" width="270"
height="237"></p>
                <p>      </p>
                <p>
                 </div>
        <p> </div>  </body>  </html>

<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-
1252">
<title>New Page 2</title>
<script language="JavaScript">
<!--
function FP_swapImg() {//v1.0
 var doc=document,args=arguments,elm,n; doc.$imgSwaps=new Array();
for(n=2; n<args.length;
 n+=2) { elm=FP_getObjectByID(args[n]); if(elm) {
doc.$imgSwaps[doc.$imgSwaps.length]=elm;
 elm.$src=elm.src; elm.src=args[n+1]; } }
}

function FP_preloadImgs() {//v1.0
 var d=document,a=arguments; if(!d.FP_imgs) d.FP_imgs=new Array();
 for(var i=0; i<a.length; i++) { d.FP_imgs[i]=new Image;
d.FP_imgs[i].src=a[i]; }
}

function FP_getObjectByID(id,o) {//v1.0
 var c,el,els,f,m,n; if(!o)o=document; if(o.getElementById)
el=o.getElementById(id);
 else if(o.layers) c=o.layers; else if(o.all) el=o.all[id]; if(el) return el;
 if(o.id==id || o.name==id) return o; if(o.childNodes) c=o.childNodes;
if(c)
 for(n=0; n<c.length; n++) { el=FP_getObjectByID(id,c[n]); if(el) return
el; }
 f=o.forms; if(f) for(n=0; n<f.length; n++) { els=f[n].elements;
 for(m=0; m<els.length; m++){ el=FP_getObjectByID(id,els[n]); if(el)
return el; } }
 return null;
}
// -->
</script>
</head>

<body onload="FP_preloadImgs(/*url*/'button26.jpg',
/*url*/'button27.jpg', /*url*/'button2F.jpg', /*url*/'button30.jpg',
/*url*/'button37.jpg', /*url*/'button38.jpg')">
```

```
<div style="border:2px solid #0099CC; position: absolute; width: 840px;
height: 468px; z-index: 1; left: 9px; top: 6px; " id="layer1">
        <div style="border:3px double #0099CC; position: absolute; width:
320px; height: 292px; z-index: 1; left: 273px; top: 134px" id="layer2">
 <p align="center">
                <a href="User.jsp">
                <img border="0" id="img1" src="button25.jpg" height="30"
width="150" alt="Random Walk" fp-style="fp-btn: Jewel 1; fp-font-style:
Bold; fp-font-color-normal: #4970B6" fp-title="Random Walk"
onmouseover="FP_swapImg(1,0,/*id*/'img1',/*url*/'button26.jpg')"
onmouseout="FP_swapImg(0,0,/*id*/'img1',/*url*/'button25.jpg')"
onmousedown="FP_swapImg(1,0,/*id*/'img1',/*url*/'button27.jpg')"
onmouseup="FP_swapImg(0,0,/*id*/'img1',/*url*/'button26.jpg')"></a><
/p>
                <p> </p>
                <p align="center">
                <a href="User2.jsp">
                <img border="0" id="img2" src="button2E.jpg" height="30"
width="150" alt="Hybrid Selection" fp-style="fp-btn: Jewel 1; fp-font-
style: Bold; fp-font-color-normal: #4970B6" fp-title="Hybrid Selection"
onmouseover="FP_swapImg(1,0,/*id*/'img2',/*url*/'button2F.jpg')"
onmouseout="FP_swapImg(0,0,/*id*/'img2',/*url*/'button2E.jpg')"
onmousedown="FP_swapImg(1,0,/*id*/'img2',/*url*/'button30.jpg')"
onmouseup="FP_swapImg(0,0,/*id*/'img2',/*url*/'button2F.jpg')"></a><
/p>
                <p> </p>
                <p align="center">
                <a href="index.jsp">
                <img border="0" id="img3" src="button36.jpg" height="30"
width="150" alt="Exit Application" fp-style="fp-btn: Jewel 1; fp-font-
style: Bold; fp-font-color-normal: #4970B6" fp-title="Exit Application"
onmouseover="FP_swapImg(1,0,/*id*/'img3',/*url*/'button37.jpg')"
onmouseout="FP_swapImg(0,0,/*id*/'img3',/*url*/'button36.jpg')"
onmousedown="FP_swapImg(1,0,/*id*/'img3',/*url*/'button38.jpg')"
onmouseup="FP_swapImg(0,0,/*id*/'img3',/*url*/'button37.jpg')"></a><
/div>
        <div style="position: absolute; width: 834px; height: 64px; z-
index: 1; left: 2px; top: 2px; background-color: #0099CC" id="layer3">
                <p align="left">
                <font
```

```
sp;         &nb
sp; 
                <font color="#FFFFFF"> 
                <b>EFFICIENT QUERY PROCESSING IN
</b></font></font>
                </p>
                <p align="left"><b><font size="6"
color="#FFFFFF">       &
nbsp;         &
nbsp;         &
nbsp;     PEER-TO-PEER
NETWORK</font></b></div>
        <p> </p>
        <p> </p>
        <p> </p>
        <div style="position: absolute; width: 100px; height: 100px; z-
index: 2; left: 59px; top: 215px" id="layer4">
                <div style="position: absolute; width: 100px; height: 100px;
z-index: 1; left: 589px; top: 3px" id="layer5">
                        <img border="0" src="bird.gif" width="140"
height="100"></div>
                <p><img border="0" src="bird.gif" width="140"
height="100"></div>
        <p> </div>

</body>

</html>

<%@ page language="java" %>
<%@ page import="java.sql.*" %>

<%!

Connection conn;
Statement st;
ResultSet res;
String tuples,tb1,tb2,tb3,tb4,tb5;
StringBuffer s=new StringBuffer();
int t,numcount;
```

```
<%
tuples=request.getParameter("name");
t=Integer.parseInt(tuples);
System.out.println("The Value T:"+t);

//Connectivity count query usage
%>

<%
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
conn=DriverManager.getConnection("jdbc:odbc:peer3","","");
st=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
res=st.executeQuery("select count (*) from sample ");
System.out.println("query executed");
res.beforeFirst();
res.last();
numcount=res.getRow();
System.out.println(numcount);

// comparing tuple value with sample size
// Sample size is less than tuple

        if(numcount<t)
        {
                System.out.println("rejected");

        }
        else
        {

        // Equal size and apply query to all .

                if(numcount==t || numcount>t)
                {

                System.out.println("apply");
```

```
                try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  conn=DriverManager.getConnection("jdbc:odbc:peer3","","");
  st=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
String Query= "SELECT TOP "+t+" sample.* FROM sample ORDER
BY Rnd(amount), sample.customerid";
res =st.executeQuery(Query);
System.out.println(" executed");

while(res.next())

{
tb1=res.getString(1);
tb2=res.getString(2);
tb3=res.getString(3);
tb4=res.getString(4);
tb5=res.getString(5);
System.out.println(tb1);
System.out.println(tb2);
System.out.println(tb3);
System.out.println(tb4);
System.out.println(tb5);
  s.append(tb1);
s.append(".");
s.append(tb2);
s.append(".");
s.append(tb3);
s.append(".");
s.append(tb4);
s.append(".");
s.append(tb5);


}
        }
        catch(Exception e)
        {
        System.out.println(e);
```

```
        }

                // Greater than the tuple size Random selection

                }

}
catch(Exception e)
{
System.out.println(e);
}

response.sendRedirect("http://localhost:8081/finalquery/result.jsp?result1
="+s.toString());
//response.sendRedirect("http://192.168.1.41:8081/finalquery/result.jsp?r
esult1="+s);

        %>
        <%!
String result1;

int counter=0;

%>
<%
result1=request.getParameter("result1");


//result1=result1.split(".");
//out.println(result1);
String[] s=result1.split("\\.");
System.out.println("length is"+s.length);
for(int i=0;i<s.length;i++)
{
counter++;
String ss1=s[i];
%>
<h4><%=ss1%></h4>
```

```
if(counter%5==0)
{
out.print("----------------------------------------------------------------------
----------");

}

}

%>
<%@ page language="java" %>
<%@ page import="java.sql.*" %>
<%!
String tuples;
Thread t=new Thread();
%>
<%

tuples=request.getParameter("D1");
if(tuples!=null)
{
System.out.println("okkkkkkkkkkkkkkkkkkkkkk ");

response.sendRedirect("http://localhost:8081/finalpeer1/index.jsp?name=
"+tuples);
 }

%>

<div style="position: absolute; width: 901px; height: 619px; z-index: 1;
left: 10px; top: 6px; border: 2px solid #0099CC" id="layer1">
        <div style="position: absolute; width: 899px; height: 64px; z-
index: 1; left: 0px; top: 0px; background-color: #0099CC" id="layer2">
                <p align="left">
                <font
size="6">        &nb
sp;         &nb
sp; 
                <font
color="#FFFFFF">       &
```

```
                <b>EFFICIENT QUERY PROCESSING IN
</b></font></font>
                </p>
                <p align="left"><b><font size="6"
color="#FFFFFF">       &
nbsp;         &
nbsp;         &
nbsp;         &
nbsp;   
                PEER-TO-PEER NETWORK</font></b></div>
        <p> </p>
        <div style="position: absolute; width: 886px; height: 368px; z-
index: 2; left: 4px; top: 104px; border: 3px double #0099CC"
id="layer3">
                        <script language="JavaScript1.2">

/*
Flying Butterfly script (By BGAudioDr@aol.com)
Modified slightly/ permission granted to Dynamic Drive to feature script
in archive
For full source, visit http://www.dynamicdrive.com
*/

var Ymax=8;                     //MAX # OF PIXEL STEPS IN THE
"X" DIRECTION
var Xmax=8;                     //MAX # OF PIXEL STEPS IN THE
"Y" DIRECTION
var Tmax=10000;                 //MAX # OF MILLISECONDS
BETWEEN PARAMETER CHANGES

//FLOATING IMAGE URLS FOR EACH IMAGE. ADD OR DELETE
ENTRIES. KEEP ELEMENT NUMERICAL ORDER STARTING
WITH "0" !!

var floatimages=new Array();
floatimages[0]='butterfly2.gif';
floatimages[1]='butterfly2.gif';
floatimages[2]='butterfly2.gif';
floatimages[3]='butterfly2.gif';
```

```
var NS4 = (navigator.appName.indexOf("Netscape")>=0 &&
parseFloat(navigator.appVersion) >= 4 &&
parseFloat(navigator.appVersion) < 5)? true : false;
var IE4 = (document.all)? true : false;
var NS6 = (parseFloat(navigator.appVersion) >= 5 &&
navigator.appName.indexOf("Netscape")>=0 )? true: false;
var wind_w, wind_h, t='', IDs=new Array();
for(i=0; i<floatimages.length; i++){
t+=(NS4)?'<layer name="pic'+i+'" visibility="hide" width="10"
height="10"><a href="javascript:hidebutterfly()">' : '<div id="pic'+i+'"
style="position:absolute; visibility:hidden;width:10px; height:10px"><a
href="javascript:hidebutterfly()">';
t+='<img src="'+floatimages[i]+'" name="p'+i+'" border="0">';
t+=(NS4)? '</a></layer>':'</a></div>';
}
document.write(t);

function moveimage(num){
if(getidleft(num)+IDs[num].W+IDs[num].Xstep >=
wind_w+getscrollx())IDs[num].Xdir=false;
if(getidleft(num)-IDs[num].Xstep<=getscrollx())IDs[num].Xdir=true;
if(getidtop(num)+IDs[num].H+IDs[num].Ystep >=
wind_h+getscrolly())IDs[num].Ydir=false;
if(getidtop(num)-IDs[num].Ystep<=getscrolly())IDs[num].Ydir=true;
moveidby(num, (IDs[num].Xdir)? IDs[num].Xstep :  -IDs[num].Xstep ,
(IDs[num].Ydir)?  IDs[num].Ystep:  -IDs[num].Ystep);
}

function getnewprops(num){
IDs[num].Ydir=Math.floor(Math.random()*2)>0;
IDs[num].Xdir=Math.floor(Math.random()*2)>0;
IDs[num].Ystep=Math.ceil(Math.random()*Ymax);
IDs[num].Xstep=Math.ceil(Math.random()*Xmax)
setTimeout('getnewprops('+num+')', Math.floor(Math.random()*Tmax));
}

function getscrollx(){
if(NS4 || NS6)return window.pageXOffset;
if(IE4)return document.body.scrollLeft;
}
```

```
function getscrolly(){
if(NS4 || NS6)return window.pageYOffset;
if(IE4)return document.body.scrollTop;
}

function getid(name){
if(NS4)return document.layers[name];
if(IE4)return document.all[name];
if(NS6)return document.getElementById(name);
}

function moveidto(num,x,y){
if(NS4)IDs[num].moveTo(x,y);
if(IE4 || NS6){
IDs[num].style.left=x+'px';
IDs[num].style.top=y+'px';
}}

function getidleft(num){
if(NS4)return IDs[num].left;
if(IE4 || NS6)return parseInt(IDs[num].style.left);
}

function getidtop(num){
if(NS4)return IDs[num].top;
if(IE4 || NS6)return parseInt(IDs[num].style.top);
}

function moveidby(num,dx,dy){
if(NS4)IDs[num].moveBy(dx, dy);
if(IE4 || NS6){
IDs[num].style.left=(getidleft(num)+dx)+'px';
IDs[num].style.top=(getidtop(num)+dy)+'px';
}}

function getwindowwidth(){
if(NS4 || NS6)return window.innerWidth;
if(IE4)return document.body.clientWidth;
}
function getwindowheight(){
```

```
if(IE4)return document.body.clientHeight;
}

function init(){
wind_w=getwindowwidth();
wind_h=getwindowheight();
for(i=0; i<floatimages.length; i++){
IDs[i]=getid('pic'+i);
if(NS4){
IDs[i].W=IDs[i].document.images["p"+i].width;
IDs[i].H=IDs[i].document.images["p"+i].height;
}
if(NS6 || IE4){
IDs[i].W=document.images["p"+i].width;
IDs[i].H=document.images["p"+i].height;
}
getnewprops(i);
moveidto(i , Math.floor(Math.random()*(wind_w-IDs[i].W)),
Math.floor(Math.random()*(wind_h-IDs[i].H)));
if(NS4)IDs[i].visibility = "show";
if(IE4 || NS6)IDs[i].style.visibility = "visible";
startfly=setInterval('moveimage('+i+')',Math.floor(Math.random()*100)+
100);
}}

function hidebutterfly(){
for(i=0; i<floatimages.length; i++){
if (IE4)
eval("document.all.pic"+i+".style.visibility='hidden'")
else if (NS6)
document.getElementById("pic"+i).style.visibility='hidden'
else if (NS4)
eval("document.pic"+i+".visibility='hide'")
clearInterval(startfly)
}
}

if (NS4||NS6||IE4){
window.onload=init;
window.onresize=function(){ wind_w=getwindowwidth();
```

```
}

</script>
            <form method="Post" action="User2.jsp">
                    <!--webbot bot="SaveResults" U-File="C:\Program
Files\Apache Software Foundation\Tomcat
5.0\webapps\ROOT\finalquery\_private\form_results.csv" S-
Format="TEXT/CSV" S-Label-Fields="TRUE" -->
                    <p> </p>
                    <p align="center"> </p>
                    <p align="center"> </p>
                    <p align="center"> </p>
                    <p
align="center">         
;           
;           
;           
;           
;           
;    </p>
                    <div style="position: absolute; width: 342px; height:
221px; z-index: 2; left: 476px; top: 125px; border: 3px double #0099CC"
id="layer5">
 <p> </p>
                        <p> </p>
                        <p align="center">
                        <input type="submit" value="Submit"
name="B1" style="color: #0099CC; font-weight: bold; border: 2px solid
#408080"></div>
                    <p align="center"> 
                    <b><font color="#0099CC"
size="4">         &nb
sp;          &nb
sp;          &nb
sp;          &nb
sp;          &nb
sp;          &nb
sp;          &nb
sp;          &nb
sp;    Number Of  Tuples
```

```
                    <select size="1" name="D1" style="color: #0099CC;
font-weight: bold; border: 2px solid #408080">
                    <option>25</option>
                    <option>50</option>
                <option>75</option>



                    </select></p>
                    <p align="center"> </p>
                    <p align="right"> </p>
                    <p> </p>
                    <div style="position: absolute; width: 309px; height:
219px; z-index: 1; left: 44px; top: 129px; border: 3px double #0099CC"
id="layer4">
                        <img border="0" src="1315054.jpg"
width="270" height="202"></div>
                    <p> </p>
                    <p> </p>
                    <p> </p>
            </form>
            <p> </div>
        <p> </div>

<%@ page language="java" %>
<%@ page import="java.sql.*" %>

<%!


Connection conn;
Statement st;
ResultSet res;
String tuples;
int t,sumamt,numcount,ycurr;
%>

<%

tuples=request.getParameter("name");
```

```
//Connectivity count query usage
%>

<%
try
{
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     conn=DriverManager.getConnection("jdbc:odbc:peer1","","");

 st=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);

 res=st.executeQuery("select count (*) from sample ");
 System.out.println("query executed");

 res.beforeFirst();

 res.last();

 numcount=res.getRow();

 System.out.println(numcount);

 // comparing tuple value with sample size


 // Sample size is less than tuple

        if(numcount<t)
        {
                System.out.println("rejected");

        }
        else
        {

        // Equal size and apply query to all .

                if(numcount==t || numcount>t)
                System.out.println("apply");


                        try
{
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conn=DriverManager.getConnection("jdbc:odbc:peer1","","");
   st=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
String Query= "SELECT sum(amount) FROM sample ";
 res =st.executeQuery(Query);
 System.out.println(" executed");

 while(res.next())

 {

 sumamt=res.getInt(1);


 }
        }
        catch(Exception e)
        {
        System.out.println(e);
        }

        }

                // Greater than the tuple size Random selection

                }

 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 System.out.println("tuples"+numcount);
```

```
System.out.println("sum"+sumamt);
ycurr=(numcount/t)*sumamt;
System.out.println("resultant value is"+ycurr);
String result1=new String();
result1=result1.valueOf(ycurr);
response.sendRedirect("http://localhost:8081/finalpeer2/index.jsp?name=
"+tuples+"&result1="+result1);

        %>

<%@ page language="java" %>
<%@ page import="java.sql.*" %>
<%!

Connection conn;
Statement st;
ResultSet res;
String tuples,result1,result2;
int t,sumamt,numcount,ycurr;
%>
<%

tuples=request.getParameter("name");
t=Integer.parseInt(tuples);
result1=request.getParameter("result1");

//Connectivity count query usage
%>

<%
try
{
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     conn=DriverManager.getConnection("jdbc:odbc:peer2","","");
st=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);

 res=st.executeQuery("select count (*) from sample ");
 System.out.println("query executed");
```

```
res.last();

numcount=res.getRow();

System.out.println(numcount);

// comparing tuple value with sample size

// Sample size is less than tuple

        if(numcount<t)
        {
                System.out.println("rejected");

        }
        else
        {

        // Equal size and apply query to all .

                if(numcount==t || numcount>t)
                {

                System.out.println("apply");


                        try
{
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   conn=DriverManager.getConnection("jdbc:odbc:peer2","","");
   st=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
String Query= "SELECT sum(amount) FROM sample ";
 res =st.executeQuery(Query);
 System.out.println(" executed");

 while(res.next())

 {
```

```
sumamt=res.getInt(1);

}
        }
        catch(Exception e)
        {
        System.out.println(e);
        }


                }

                // Greater than the tuple size Random selection

        }

}
catch(Exception e)
{
System.out.println(e);
}
System.out.println("tuples"+numcount);
System.out.println("processedtuples"+t);
System.out.println("sum"+sumamt);
ycurr=(numcount/t)*sumamt;
System.out.println("resultant value is"+ycurr);
result2=result2.valueOf(ycurr);
response.sendRedirect("http://localhost:8081/finalquery/result1.jsp?result
1="+result1+"&result2="+result2);

        %>
                <%!
String result1,result2;
int resultp1,resultp2;
float prob,deg,edge,y1,y2,error;

String s1=new String();
String s2=new String();
String e=new String();
```

```
<%
result1=request.getParameter("result1");
result2=request.getParameter("result2");
//System.//out.println("result1==="+result1);
//System.//out.println("result2==="+result2);

resultp1=Integer.parseInt(result1);
resultp2=Integer.parseInt(result2);

deg=3;
edge=8;
prob=deg/edge;
//out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"+prob);

y1=((float)resultp1/prob)/(3/2);
//out.println("The Y1 Value:"+y1);
y2=((float)resultp2/prob)/(3/2);
//out.println("The Y2 Value:"+y2);

error=(y1-y2);
///System.out.println("The Error Rate:"+error);
int r=Math.round(y2);
//out.println(r);

s1=s1.valueOf(y1);
s2=s2.valueOf(y2);
e=e.valueOf(error);
%>
<div style="position: absolute; width: 901px; height: 527px; z-index: 1;
left: 47px; top: 33px; border-style: solid; border-width: 1px"
id="layer1">
                <script language="JavaScript1.2">

/*
Flying Butterfly script (By BGAudioDr@aol.com)
Modified slightly/ permission granted to Dynamic Drive to feature script
in archive
For full source, visit http://www.dynamicdrive.com
*/
```

```
var Ymax=8;                     //MAX # OF PIXEL STEPS IN THE
"X" DIRECTION
var Xmax=8;                     //MAX # OF PIXEL STEPS IN THE
"Y" DIRECTION
var Tmax=10000;                 //MAX # OF MILLISECONDS
BETWEEN PARAMETER CHANGES
//FLOATING IMAGE URLS FOR EACH IMAGE. ADD OR DELETE
ENTRIES. KEEP ELEMENT NUMERICAL ORDER STARTING
WITH "0" !!
var floatimages=new Array();
floatimages[0]='butterfly2.gif';
floatimages[1]='butterfly2.gif';
floatimages[2]='butterfly2.gif';
floatimages[3]='butterfly2.gif';

//*********DO NOT EDIT BELOW**********
var NS4 = (navigator.appName.indexOf("Netscape")>=0 &&
parseFloat(navigator.appVersion) >= 4 &&
parseFloat(navigator.appVersion) < 5)? true : false;
var IE4 = (document.all)? true : false;
var NS6 = (parseFloat(navigator.appVersion) >= 5 &&
navigator.appName.indexOf("Netscape")>=0 )? true: false;
var wind_w, wind_h, t='', IDs=new Array();
for(i=0; i<floatimages.length; i++){
t+=(NS4)?'<layer name="pic'+i+'" visibility="hide" width="10"
height="10"><a href="javascript:hidebutterfly()">' : '<div id="pic'+i+'"
style="position:absolute; visibility:hidden;width:10px; height:10px"><a
href="javascript:hidebutterfly()">';
t+='<img src="'+floatimages[i]+'" name="p'+i+'" border="0">';
t+=(NS4)? '</a></layer>':'</a></div>';
}
document.write(t);

function moveimage(num){
if(getidleft(num)+IDs[num].W+IDs[num].Xstep >=
wind_w+getscrollx())IDs[num].Xdir=false;
if(getidleft(num)-IDs[num].Xstep<=getscrollx())IDs[num].Xdir=true;
if(getidtop(num)+IDs[num].H+IDs[num].Ystep >=
wind_h+getscrolly())IDs[num].Ydir=false;
if(getidtop(num)-IDs[num].Ystep<=getscrolly())IDs[num].Ydir=true;
```

```
moveidby(num, (IDs[num].Xdir)? IDs[num].Xstep :  -IDs[num].Xstep ,
(IDs[num].Ydir)?  IDs[num].Ystep:  -IDs[num].Ystep);
}

function getnewprops(num){
IDs[num].Ydir=Math.floor(Math.random()*2)>0;
IDs[num].Xdir=Math.floor(Math.random()*2)>0;
IDs[num].Ystep=Math.ceil(Math.random()*Ymax);
IDs[num].Xstep=Math.ceil(Math.random()*Xmax)
setTimeout('getnewprops('+num+')', Math.floor(Math.random()*Tmax));
}

function getscrollx(){
if(NS4 || NS6)return window.pageXOffset;
if(IE4)return document.body.scrollLeft;
}

function getscrolly(){
if(NS4 || NS6)return window.pageYOffset;
if(IE4)return document.body.scrollTop;
}

function getid(name){
if(NS4)return document.layers[name];
if(IE4)return document.all[name];
if(NS6)return document.getElementById(name);
}

function moveidto(num,x,y){
if(NS4)IDs[num].moveTo(x,y);
if(IE4 || NS6){
IDs[num].style.left=x+'px';
IDs[num].style.top=y+'px';
}}

function getidleft(num){
if(NS4)return IDs[num].left;
if(IE4 || NS6)return parseInt(IDs[num].style.left);
}
```

```
if(NS4)return IDs[num].top;
if(IE4 || NS6)return parseInt(IDs[num].style.top);
}

function moveidby(num,dx,dy){
if(NS4)IDs[num].moveBy(dx, dy);
if(IE4 || NS6){
IDs[num].style.left=(getidleft(num)+dx)+'px';
IDs[num].style.top=(getidtop(num)+dy)+'px';
}}

function getwindowwidth(){
if(NS4 || NS6)return window.innerWidth;
if(IE4)return document.body.clientWidth;
}

function getwindowheight(){
if(NS4 || NS6)return window.innerHeight;
if(IE4)return document.body.clientHeight;
}

function init(){
wind_w=getwindowwidth();
wind_h=getwindowheight();
for(i=0; i<floatimages.length; i++){
IDs[i]=getid('pic'+i);
if(NS4){
IDs[i].W=IDs[i].document.images["p"+i].width;
IDs[i].H=IDs[i].document.images["p"+i].height;
}
if(NS6 || IE4){
IDs[i].W=document.images["p"+i].width;
IDs[i].H=document.images["p"+i].height;
}
getnewprops(i);
moveidto(i , Math.floor(Math.random()*(wind_w-IDs[i].W)),
Math.floor(Math.random()*(wind_h-IDs[i].H)));
if(NS4)IDs[i].visibility = "show";
if(IE4 || NS6)IDs[i].style.visibility = "visible";
startfly=setInterval('moveimage('+i+')',Math.floor(Math.random()*100)+
```

```
}}

function hidebutterfly(){
for(i=0; i<floatimages.length; i++){
if (IE4)
eval("document.all.pic"+i+".style.visibility='hidden'")
else if (NS6)
document.getElementById("pic"+i).style.visibility='hidden'
else if (NS4)
eval("document.pic"+i+".visibility='hide'")
clearInterval(startfly)
}
}

if (NS4||NS6||IE4){
window.onload=init;
window.onresize=function(){ wind_w=getwindowwidth();
wind_h=getwindowheight(); }
}

</script>
        <div style="position: absolute; width: 897px; height: 86px; z-
index: 1; left: 2px; top: 2px; background-color: #0099CC" id="layer2">
                <p align="center"><font color="#FFFFFF"
size="6">   EFFICIENT QUERY PROCESSING IN
</font>
                </p>
                <p align="center"><font size="6"
color="#FFFFFF">  PEER-TO-PEER NETWORK</font></div>
        <div style="position: absolute; width: 897px; height: 29px; z-
index: 2; left: 0px; top: 495px; background-color: #0099CC"
id="layer3">
 </div>
        <div style="position: absolute; width: 231px; height: 388px; z-
index: 4; left: 39px; top: 104px" id="layer5">
                <div style="position: absolute; width: 33px; height: 433px;
z-index: 1; left: -36px; top: -13px; background-color: #0099CC"
id="layer6">
<div style="position: absolute; width: 331px; height: 321px; z-index: 1;
left: 388px; top: 35px" id="layer8">
```

```
                <!--webbot bot="SaveResults" U-
File="fpweb:///_private/form_results.csv" S-Format="TEXT/CSV" S-
Label-Fields="TRUE" -->
                <p> </p>
                <p> </p>
                <p><b><font face="Century Schoolbook" color="#0099CC"
size="2">Result
                From System 1: </font></b><font color="#0099CC"
size="3" face="Century Schoolbook">
                <input name="T1" size="20"  value="<%=s1%>"
disabled="false"  style="border:1px solid #0066CC; font-weight:
700"></font></p>
                <p><b><font face="Century Schoolbook" color="#0099CC"
size="2">Result
                From System 2: </font></b><font color="#0099CC"
size="3" face="Century Schoolbook">
<input name="T2" size="20"  value="<%=s2%>" disabled="false"
style="border:1px solid #0066FF; font-weight: 700"></font></p>
                <p><b><font face="Century Schoolbook" color="#0099CC"
size="2">Error
        Rate        &n
bsp;         &n
bsp;
                : </font></b><font color="#0099CC" size="3"
face="Century Schoolbook">
                <input name="T3" size="20"  value="<%=e%>"
disabled="false" style="border:1px solid #0066FF; font-weight:
700"></font></p>
        </form>
        <p> </div>
 </div>
                <p> </p>
                <p> </p>
                <p>
                <img border="0" src="874256.jpg" width="270"
height="237"></p>
                <p>      </p></div>
        <p> </div>

<h5> </h5>
```

CHAPTER 9

REFERENCES

BOOKS AND JOURNALS:

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

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

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

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

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

6. **http://www.j2ee.com**

7. **http://www.apache.com**