IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Aggregate Query Processing using Random walk approach in Dynamic Environment

Vinod S. Gangwani[*1], Prof. P. L. Ramteke[2]

[*1]Department of Computer Science & Engineering H.V.P.M's COET, SGBAU, Amravati (MH), India

[2]Department of Information Technology H.V.P.M's COET, SGBAU, Amravati (MH), India

Vinod.gangwani@gmail.com

Abstract

Peer-to-peer (P2P) network is increasingly becoming popular because it offers oppor¬tunities for real-time communication, ad-hoc collaboration and information sharing in a large-scale distributed environment. Peer-to-peer computing is defined as the sharing of computer resources and information through direct exchangeThe advantages of the P2P systems are multi-dimensional; they improve scalability by enabling direct and real-time sharing of services and information; enable knowledge sharing by aggregating information and resources from nodes that are located on geographically distributed and potentially heterogeneous platforms; and, provide high availability by eliminating the need for a single centralized component.

The problem of answering large scale, ad-hoc analysis queries – e.g., 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 peer-to-peer databases. In this paper we present novel sampling-based techniques for approximate answering of ad-hoc aggregation queries in such databases. 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.

Keywords : Random Walk, Query Processing

Introduction

Peer-to-Peer Databases: The peer-to-peer network model is quickly becoming the preferred medium for file sharing and distributing data over the Internet. A peer-to peer (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 network 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 [25], Gnutella [15], KaZaA [20] and Freenet [13] are principally known for their file sharing capabilities, e.g., the sharing of songs, music, and so on. Furthermore, researchers have been interested in extending sophisticated IR techniques such as keyword search and relevance retrieval to P2P databases.

Aggregation Queries:

In this paper, however, we consider a problem on P2P systems that is different from the typical search and retrieval applications. As P2P systems mature beyond file sharing applications and start getting deployed in increasingly sophisticated e-business and scientific environments, the vast amount of data within P2P databases pose a different challenge that has not been adequately researched thus far – that of how to answer aggregation queries on such databases. 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 interesting in asking decision support queries that require the aggregation of vast amounts of data covering thousands of peers. An aggregation query such as the following may be introduced at any peer (this peer is henceforth called the sink)

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. While our main focus is on the above standard SQL aggregation operators.

Approximate Query Processing

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 the answer can be obtained fast enough. This observation has been the primary driving force behind recent development of approximate query processing (AQP) techniques for aggregation queries in traditional databases and decision support systems [9, 3,6, 8, 1, 14, 5, 7, 23]. Numerous AQP techniques have been developed, the most popular ones 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 extrapolated to the whole database.

Goal of Paper: Approximating Aggregation Queries in P2P Networks

Given an aggregation query and a desired error bound at a sink 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 straight forward 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, e.g., the number of participating peers, the bandwidth consumed (i.e., amount of data shipped over the network), the number of messages exchanged, the latency (the end-to-end time to propagate the query across multiple peers and receive replies), the I/O cost of accessing data from participating peers, the CPU cost of processing data at participating peers, and so on. In this paper, we shall be concerned with several of these cost metrics.

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 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, Our approach has two major phases. In the first phase, we initiate a fixed-length random walk from the sink. This random walk should be long enough to ensure that the visited peers1 represent a close sample from the underlying stationary distribution – the appropriate length of such a walk is determined in a pre-processing step. We then retrieve certain information from the visited peers, such as the number of tuples, the aggregate of tuples (e.g., SUM/COUNT/AVG, etc.) that satisfy the selection condition, and send this information back to the sink. This information is then analyzed at the sink 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 and so on. Once this data has been analyzed at the sink, an estimation is made on how much more samples are required - and in what way should these samples be collected - so that the original query can be optimally answered within the desired accuracy with high probability. For example, the first phase may recommend that the best way to answer this query is to visit m' more peers, and from each peer, randomly sample t tuples. We mention that the first phase is not overly driven by heuristics – instead it is based on strong underlying theoretical principles, such as theory of random walks [14, 21, 4], as well as statistical techniques such as cluster sampling, block-level sampling and cross validation [9, 16]. The second phase is then straightforward – a random walk is reinitiated and tuples collected according to the recommendations made by the first phase. Effectively, the first phase is used to "sniff" the network and determine an optimal-cost "query plan", which is then implemented in the second phase. For certain aggregates, such as COUNT and SUM, further optimizations may be achieved by pushing the selections and aggregations to the peers – i.e., the local aggregates instead of raw samples are returned to the sink, which are then composed into a final answer.

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

Query Cost Measures

As mentioned in the introduction, the cost of the execution of a query in P2P databases is more complicated that equivalent cost measures in traditional databases. The primary cost measure we consider is latency, which is the end-to-end time to propagate the query across multiple peers and receive replies. For the purpose of illustration, we focus in this section on the SUM and COUNT aggregates. For these specific aggregates, latency can be approximated by an even simpler measure: the number of peers that participate in the algorithm. This measure is appropriate for these aggregates primarily because the overheads of visiting peers dominate other incurred costs. Random Walk in Graphs.

Random Walk in Graphs :

In seeking a random sample of the P2P database, we have to overcome the sub-problem of how to collect a random sample of the peers themselves. Unrepresentative samples of peers can quickly skew results producing erroneous aggregation statistics. Sampling in a non-hierarchical decentralized P2P network presents several obstacles in obtaining near uniform random samples. This is because no peer (including the query sink) 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, clearly the sink could locally generate a random subset of IP addresses from among all the IP addresses, and visit the appropriate peers directly. This problem has been recognized in other contexts 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 sink, and for each visited peer, the next peer to visit is selected with equal probability from among its neighbors (and itself – thus self loops are allowed). It is well known that, if this walk is carried out long enough, 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(p). Then the probability of any peer p in the stationary distribution is

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

Our Algorithm

In this section we present details of our two-phase algorithm for approximating answering of aggregate queries. For the sake of illustration, we focus on approximating COUNT queries – it can be easily extended to SUM queries. The pseudo code of the algorithm is presented below.

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 I
j : Jump size for random walk
t : Max #tuples to be sub-sampled per peer
Inputs
Q : COUNT query with selection condition
Sink : Peer where query is initiated
req _ : Desired max error
Phase I
// Perform Random Walk
1. Curr = Sink; Hops = 1;
2. while (Hops < j * m) {

3. if (Hops % j)
4. Visit(Curr);
5. Hops++;
6. Curr = random adjacent peer
7. }

// Visit Peer

1. Visit(Curr) {
2. if (#tuples of Curr) <= t) {
3. Execute Q on all tuples
4. else
5. Execute Q on t randomly sampled
6. tuples
7. }
8.

$$y(curr) = \left(\frac{\#tuples}{\#processedTuples}\right) * (resul\_of\_Q)$$

10. Return (y(Curr), deg(Curr)) to Sink
11. }

// Cross-Validate at Sink

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

$$y_1^{"} = \frac{\sum_{s \in s_1} y(s)/prob(s)}{m/2}$$

$$y_2^{"} = \frac{\sum_{s \in s_2} y(s)/prob(s)}{m/2}$$

Where

$$prob(p) = \frac{dep(p)}{2E}$$

1. Visit m' peers using random walk
2. Let S' = {s1, s2, …, sm'} be the visited peers

Our approach in the first phase is broken up into the following main components. First, we perform a random walk on the peer-to-peer 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 sink. Note that in the previous section, 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 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; and for such cost models, an ideal two-phase algorithm should determine various parameters in the first phase, such as how many peers to visit in the second phase, and how many tuples to sub-sample from each visited peer. In this paper we 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, 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 – e.g., 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, 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 cross-validation error 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 and 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. Although the algorithm has been presented for the case of COUNT, it can be easily extended for SUM. Finally, we re-emphasize that for more complex aggregates, such as estimation of medians, quintiles, and distinct values, more sophisticated algorithms are required.

Conclusion

In this paper we present adaptive sampling-based techniques for the novel problem of approximate answering of ad-hoc aggregation queries in P2P databases. We present extensive experimental evaluations to demonstrate the feasibility of our solutions. Several intriguing open problems remain. Is it possible to build hybrid solutions that do some amount of pre-computations of samples, in addition to "on-the-fly" sampling such as ours? Is it possible

for sampling-based algorithms to perform "biased sampling", i.e., focus the samples from regions of the database where tuples that satisfy the query are likely to exist? More generally, decision support and data analysis in P2P databases appears to be an important area of research with emerging applications, and we hope our work will encourage further research in this field.

References

[1] S. Acharya, P. B. Gibbons and V. Poosala. Aqua: A Fast Decision Support System Using Approximate Query Answers. Demo in Intl. Conf. on Very Large Databases (VLDB '99).

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

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

[4] A.R. Bharambe, M. Agrawal, and S. Seshan. Mercury:Supporting Scalable Multi-Attribute Range Queries. SIGCOMM 2004.

[5] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In Proceedings of the ACM Symp. On Principles of Database Systems, 2000.

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

[7] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? IN Proceedings. Of the 1998 ACM SIGMOD Intl. Conf. on Management of Data, pages 436-447, 1998.

[8] S. Chaudhuri, G. Das, and V. Narasayya. A Robust, Optimization-Based Approach for Approximate Answering of Aggregate Queries. SIGMOD Conference 2001.

[9] S. Chaudhuri, G. Das, and U. Srivastava. Effective Use of Block-Level Sampling in Statistics Estimation. SIGMOD 2004.

[10] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In Proceedings of ACM Sigmetrics 2000.

[11] Mauricio Minuto Espil and Alejandro A. Vaisman. Aggregate queries in peer-to-peer OLAP. DOLAP '04.

[12] C. Faloutsos, P. Faloutsos, and M. Faloutsos. On Power- Law Relationships of the Internet Topology. SIGCOMM 1999.

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

[14] C. Gkantsidis, M. Mihail, and A. Saberi. Random Walks in Peer-to-Peer Networks. IEEE Infocom 2004.

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

[16] P. Haas, and C. K[nig. A Bi-Level Bernoulli Scheme for Database Sampling. SIGMOD 2004.

[17] R. Heubsch, J. Hellerstein, N. Lanhan, B. T. Loo, S.Shenker, and I. Stoica. Querying the Internet with PIER. VLDB 2003.

[18] JUNG website. http://jung.sourceforge.net.

[19] P. Kalnis, W. S. Ng, B. C. Ooi and D. Papadias and K-L.Tan. An adaptive peer-to-peer network for distributed caching of OLAP results. SIGMOD 2002.

[20] KaZaA Homepage, http://www.kazaa.com.

[21] V. King and J. Saia. Choosing a Random Peer. PODC 2004.

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

[23] X. Li, Y.J. Kim, R. Govindan, and W. Hong. Multidimensional range queries in sensor networks. SENSYS 2003.

[24] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J.Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. HP Technical Report, HPL-2002-57.

[25] Napster Hompage, http://www.napster.com.

[26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S.Shenker. A Scalable Content-Addressable Network. SIGCOMM 2001.

[27] A. Rowstron and P. Druschel. Pastry: Scalable,distributed object location and routing for large-scale peer-topeer systems. IFIP/ACM Middleware 2001.