Dynamic Query Plan for Efficient Query Processing in Peer-to-Peer Environments

MD MEHEDI MASUD[1]
M. ANWAR HOSSAIN[2]

University of Ottawa
School of Information Technology and Engineering (SITE)
800 King Edward St., K1N 6N5, Ottawa, Ontario,Canada
[1]mmasud@site.uottawa.ca
[2]anwar@mcrlab.uottawa.ca

Abstract. In the past few years, Peer-to-Peer (P2P) applications have emerged as a popular way of autonomously sharing data and services in distributed environments. In such environments, peers dynamically join/leave a network and do not usually have any global knowledge about the data sources. This phenomenon demands an efficient query execution strategy in terms of data transmission and response time. In this paper, we propose a distributed query processing technique in P2P environments where each peer possesses partial knowledge of the domain information and collaborates with the participating peers to share data. The cornerstone of our approach is to dynamically determine the best query execution plan in order to execute the different parts of the user query, and propagate the results with minimized data transmission and response time.

Keywords: Peer-to-peer, Information retrieval, Query processing.

(Received January 24, 2008 / Accepted July 16, 2008)

1 Introduction

In the past few years, the P2P technology has emerged as a new paradigm for distributed data sharing systems. In this technology, all participating peers have equivalent capabilities and responsibilities, and exchange resources (e.g. data) and services (e.g. query processing) through pair-wise communication. From the point of view of distribution, a P2P system resembles the traditional distributed system. However, the solutions for data sharing and access can not be directly applied to a P2P system. This is mainly due to the three distinct features of the P2P paradigm: the lack of centralized control; the transience of the inter-peer connections; the limited cooperation among the peers. Therefore, from the point of view of distribution, different techniques are used for P2P data sharing and query processing.

Today, the popular P2P infrastructure enables us to store and share data in distributed and decentralized fashion. Until now, many P2P systems (such as Napster [4], Gnutella [6], MUTE [1] etc.), and frameworks (such as JXTA [2], Jabber [3] etc.) have emerged. In the P2P paradigm, each peer in the network manages their data autonomously and collaborates with the neighboring peers to access or share data. However, a peer has no global knowledge about the resources possessed by all the peers in the network, it only has the knowledge about its acquainted neighbors.

The lack of global knowledge about the data poses challenge for any peer to execute a query over the P2P network. Unlike the traditional distributed systems, the P2P model does not employ any centralized coordinator to process user query. However, the basic distributed query processing strategies [19] such as *data shipping* and *query shipping* are adopted in P2P systems. In data shipping, all the peers move the data to the query initiator and all operations are executed at the initiator. In

query shipping, the query is moved to the peers to be evaluated by them. If a peer's data satisfies a part of a query, the peer sends the matching data to the requester for further processing. This approach avoids unnecessary data shipping in the network and reduces network traffic.

In P2P, a user query is decomposed into sub-queries according to the execution capabilities of the peers. Therefore, a query is partially evaluated at one peer, and is propagated to another peer until it is fully evaluated. However, the propagation and execution of a query/sub-query is influenced by the current acquaintances, network traffic, local knowledge about data, execution cost, and data transmission cost. These factors need to be considered for dynamically planning query execution path and determining whether to use data shipping, query shipping, or hybrid of the both query processing techniques.

In some well known approaches [16], [14], [15] that deal with P2P query processing, a user query is shipped with the intermediate results from one peer to another peer. These approaches cause higher network traffic due to the shipment of intermediate results along with the query plan. In addition, it incurs increased response time as the target peer has to wait for a long time to get the final result depending on the network condition and the volume of the data. Therefore, we need an efficient query propagation and execution plan that dynamically considers the up-to-date cost information in a P2P system. In this paper, we contribute in this direction. Our contribution is mainly twofold:

- First, we present an improved query processing technique for generating dynamic query plan. The technique considers network path and processing cost of intermediate results based on data distribution on different peers and query operators.

- Second, we performed a comparison in terms of query execution cost and show the efficiency of the proposed technique.

The remaining of this paper is organized as follows. Section 2 comments on some related works. A motivating example is presented in Section 3. The proposed methodology is described in Section 4. In Section 5, we state the evaluation results of our approach, and finally, the paper is concluded in Section 6.

2 Related Study

There are many domain specific P2P systems that have already been developed. Among these, Gnutella [6] and KaZaA [5] are two most celebrated file sharing applications, which use some kind of centralized indexing techniques for providing search operations. To find data in more efficient way, a class of P2P systems have been developed by structuring the data so that it can be found with far less expenses than traditional flooding-based approach. This technique is commonly referred to as Distributed Hash Tables (DHTs) [23, 7, 8, 9]. DHTs are well-suited for exact match lookups using unique identifiers, but do not directly support text search. Some other well-known file sharing systems such as [1], [21], and [22] use ant-inspired or similar routing protocol for its search functions.

The above approaches provide simple IR-style string matching or meta-data matching techniques for searching files in peers. However, they lack advanced database-style data management, query processing, and other functionalities. The database-style data manipulation in P2P is more robust than the conventional file sharing approach where peers are involved in processing and shipping intermediate query results in order to produce the final results.

There are some research on P2P systems that deal with the database management issues [10, 11, 12, 20]. They mainly consider data models for P2P databases, peer schema mediation methods, coordination mechanisms between peer databases, and mapping data among acquainted peers. Authors in [13], propose a query translation mechanism between heterogeneous peers considering instance-level mappings. However, they did not consider the efficient way of query processing that reduces query processing time to produce the final results.

In this paper, we present a query processing strategy in distributed data sharing P2P systems. Similar work has been done in Mutant Query Plan (MQP) [16], [17]. However, unlike ours, MQP does not consider query decomposition and dynamic execution plan based on cost of intermediate data processing, cost of data transmission, and the up-to-date information of the acquaintances among peers. Moreover, MQP always uses the fixed hybrid techniques (i.e. query shipping and data shipping), our proposed model dynamically decides whether to use query shipping, data shipping or the hybrid approach depending on the query and result. Although our approach introduces some additional (but minimal) message transfers over the network, it reduces the overall cost of query processing in the distributed P2P environments.

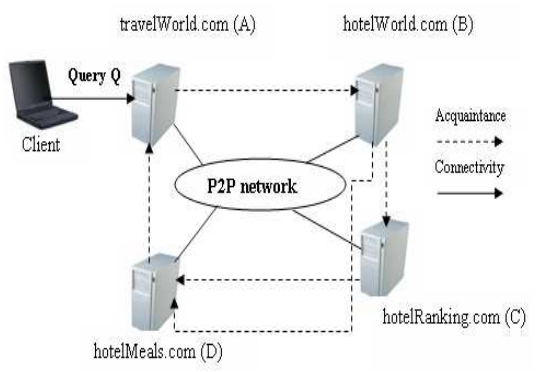

Figure 1: A motivating scenario

3 Motivating Example

In this section, we present a simplified real-world scenario to show the applicability of the proposed query processing technique in a P2P environment. We consider an example from the world-wide travel domain that helps a tourist to find suitable resort of his/her preferences. The whole scenario is depicted in Figure 1. In this scenario, peers create a virtual network with their acquaintances, exchange data among them, and take part to execute a query or part of a query posed on any peer.

Consider that the peer A is a portal site $travelWorld.com$, which provides access of information about the various resorts over the world. Peer B stores the information of hotels, which can be accessed through the site $hotelWorld.com$. Peer C is the site $rankingHotel.com$ that provides the ranking of different hotels. Finally, peer D provides information about meals of different hotels which is accessed via $hotelMeals.com$. In the context of our system, we assume a peer offers some web services to deliver its query functionality and contains data in XML documents. Each peer consists of an XML search engine which support XQuery-based queries.

Now, consider a case where a user wants to travel to a country for his summer vacation. He wants to find hotels which best matches his preferences. Assume that the user visits his favorite portal, travelWorld.com, where he asks the query service to get best hotel information. Consider the user submits the following query.

FOR $\$h$ in document ("hotel"),
$\$p$ in document("preference"),
$\$m$ in document("meals") [meal="vegetarian"],
$\$r$ in document("hotel ranking") [rank>=1 AND rank<=10] WHERE $\$h$ = $\$p$ AND $\$h$/name = $\$r$/name;
AND $\$h$/name = $\$m$/name;
RETURN $< hotel >$ $\{\$h/name\}$ $\{\$h/address\}$ $\{\$r/ranking\}$ $< /hotel >$

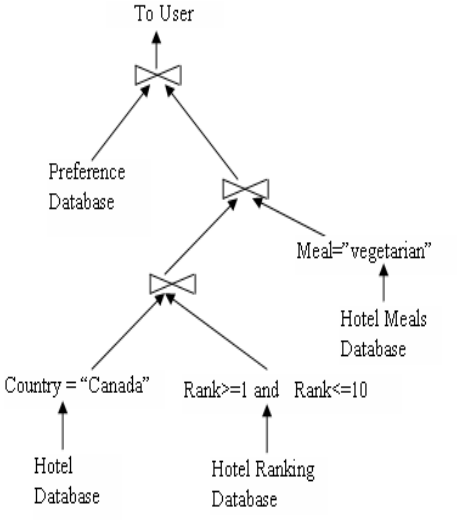

Figure 2: Initial query plan

The query processor at the portal site translates the query into a logical query plan as shown in Figure 2. From the query plan, it is obvious that the query needs to access four peers to get the final results. In this scenario, we need an efficient query processing strategy among peers that gives the minimized execution cost of the query in terms of the data transmission cost and the response time. In the next section, we describe such a strategy.

4 Proposed query processing method

The basic distributed query processing strategies are data shipping and query shipping. In this work, we are motivated by a strategy called Mutant Query Plan (MQP) [16], which uses a combination of data shipping and query shipping mechanism. In MQP, first a regular query operator tree is formed at the query initiator and then the trees is passed to the next peer. During passing of the query operator tree, the peers accumulate partial results. The passing of the query with partial results continues from one peer to another peer until it is fully evaluated into a constant piece of XML data and returned to the query initiator. This approach shows inefficiency when we have large number of peers and those peers are involved in processing the query. One of the

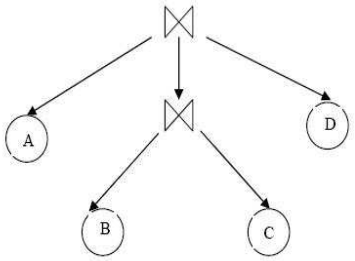

Figure 3: The execution plan of the query

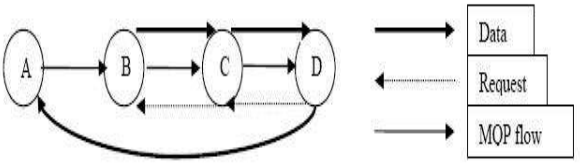

Figure 4: State diagram for the flow of query and data

reasons is that the size of the MQP may be too large and hence, shipping the MQP with data from one peer to another becomes expensive.

In our approach, we provide a dynamic query execution strategy that may involve data shipping, the query shipping, or the query with data like in MQP, which completely depends on the intermediate results of the query in a peer. When a query is initiated, the local peer generates a query execution plan based on the local information of its acquaintances. The local peer then evaluates the query and cache the partial results. Before passing the query to its acquaintances for further execution, a peer exchanges some messages with its acquaintances, and takes decision about the prospective execution plan. There are three types of messages a peer uses to communicate with its acquaintances. Before describing different message primitives, we first shortly describe different notations that are used to represent a message.

- Q_i, a query initiated at peer P_i
- C_{Qi}^{Pi}, cardinality of the intermediate results of query Q_i at peer P_i. This defines the number of rows in the result.
- R_{Qi}^{Pi}, intermediate result of the query Q_i at peer P_i
- N_i, the network address of a peer P_i

Using the above notations, the different message primitives are described as follows:

1. *Push query* $< Q_i, C_{Qi}^{Pi}, N_i >$ - A peer P_i which receives a query Q_i, first executes the part of the query that satisfies its data and the results are stored in a temporary cache. Then it forwards the query Q_i to its acquaintances P_j, where the query needs to execute further. On forwarding the query, the peer also attaches the network address N_i and the cardinality C_{Qi}^{Pi} of the intermediate result processed at the peer locally.

2. *Pull data* $< Q_i, N_j >$ - A peer P_j which receives a message $< Q_i, C_{Qi}^{Pi}, N_i >$, first executes the part of the query which satisfies its data. After evaluating local result, the peer performs two tasks. First it checks whether it requires to process the query further with the result of the peer P_i. If it needs to perform further processing, then it compares the cardinality C_{Qi}^{Pj} with cardinality C_{Qi}^{Pi}. If $C_{Qi}^{Pj} > C_{Qi}^{Pi}$, then it sends the message *Pull data* $< Q_i, N_j >$ to peer P_i. Otherwise, it sends *Push data* $< Q_i, R_{Q_i}^{P_j}, N_j, N_k >$. If there is no operator involved to process results of P_i and P_j, then P_j sends *Push query* $< Q_i, C_{Qi}^{Pj}, N_j >$ to its acquaintance P_k for further processing of the query.

3. *Push data* $< Q_i, R_{Q_i}^{P_j}, N_j, N_k >$ - When a peer P_i receives a *Push data* $< Q_i, R_{Q_i}^{P_j}, N_j, N_k >$ message from P_j, it then realizes that results of $R_{Q_i}^{P_j}$ and $R_{Q_i}^{P_i}$ needs to be processed further. This scenario happens when the volume of intermediate results of P_j is larger than P_i. After processing the intermediated results, P_i then sends a *Push query* $< Q_i, C_{Qi}^{Pi}, N_i >$ to peer P_k where the query should be executed further. Note that the P_k is not an acquaintance of P_i, but an acquaintance of P_j. P_i knows the address of P_k through the message *Push data* $< Q_i, R_{Q_i}^{P_j}, N_j, N_k >$.

Instead of shipping both data and query, our approach ships data from one peer to another. The cardinality of the message is used for the optimization purpose to take decision which peer will send data. To illustrate, suppose a peer *A* evaluates a sub-plan of a query. Instead of attaching data like MQP, in our approach, peer A sends *push* query message to its acquaintance which includes the network address of peer A and the cardinality of the intermediate results. When peer B receives the message, it then evaluates the subplan which satisfies the query. If the results of B need to process with the results of peer A, then either B re-

Query execution sequence	Transmission (Data + Query)	Cost (Data + Message)	Descriptions
$A \rightarrow B$	200 + MQP	200 1	A sends MQP with data to B
$B \rightarrow C$	200+100+ MQP	300 1	B sends MQP with data to C
$C \rightarrow D$	100x10+200= 300 + MQP	300 1	C process the operation and sends MQP with data to D
$D \rightarrow A$	200x100x200= 100	100	D process the whole query and send result to A
Total		900 4	

Table 1: Query execution cost with MQP

Query execution sequence	Transmission (Data + Query)	Cost (Data + Message)	Descriptions
$A \rightarrow B$	0 + Push Query	0 1	A sends query to C
$B \rightarrow C$	0 + Push Query	0 1	B sends query to C
$C \rightarrow B$	0 + Pull Data	0 1	C request B for Data
$B \rightarrow C$	100+ Push Data	100 1	B sends data to C
$C \rightarrow D$	0+ Push query	0 1	C process operation and sends Query to D
$D \rightarrow C$ $D \rightarrow A$	0+ Pull Data	0 1	D request data from A and C
$C \rightarrow D$ $A \rightarrow D$	200 100 Push Data	300 2	A and C send data to D
$D \rightarrow A$	100 Push Data	100 1	D sends data to A
Total Cost		500 7	

Table 2: Query execution cost with the proposed approach

quests A with *pull* message to send its result or B sends its result with $push$ data message to A for further processing of the query. In this way, the strategy reduces the overall data transmission cost.

In the following, we present a small example in order to illustrate the procedure of query processing and routing, by considering the scenario described in Section 3. For ease of presentation, we denote peer *travelWorld.com* as A, *hotelWorld.com* as B, *hotelRanking.com* as C, and *hotelMeals.com* as D. Suppose a query Q_A is submitted to the portal site of A. Peer A then produces a constant XML fragment $R^A_{Q_A}$ of user's preference. The XML document is temporarily is stored in the local cache. Now, A is unable to do any further evaluation of the query because it has no information about hotels, ranking of hotels, and meal preferences of hotels in its local database. But A knows that the acquaintance peer B can provide detail information about hotels. So A forwards a $push$ query message $< Q_A, C^A_{Q_A}, N_A >$ to B. Peer B then evaluates the query partially that satisfies its data, and produces results $R^B_{Q_A}$ as another XML fragment which is stored in the cache. Peer B has no ranking information. Therefore it sends a $push$ query message $< Q_A, C^B_{Q_A}, N_B >$ to C, where peer C is an acquaintance of B.

At this moment, C realizes that the query needs to be processed further which involves join operation of $R^B_{Q_A}$ and $R^C_{Q_A}$. Therefore, C determines the cost of efficient execution of the operator. Fortunately, C knows the cardinality of $R^B_{Q_A}$ which C receives as a $push$ query message $< Q_A, C^B_{Q_A}, N_B >$ from B. Therefore, C compares the cardinality of $R^B_{Q_A}$ and $R^C_{Q_A}$. In this case, we assume that $C^B_{Q_A} > C^C_{Q_A}$. Hence, C sends a *pull* data message $< Q_A, N_C >$. After receiving the message, B sends a *push* data message $< Q_A, R^B_{Q_A}, N_B, null >$. Peer C now process the operation and stores the result in the cache. Peer C notices that the query Q_A requires further processing, but it has no information of meal preference of different hotels. It knows that D has the information, thus sends a push query message $< Q_A, C^C_{Q_A}, N_C >$ to D.

After receiving the message from C, D then evaluates the query and makes a plan to evaluate the query further. We assume that $C^D_{Q_A} > C^C_{Q_A}$ and $C^D_{Q_A} > C^A_{Q_A}$. Therefore, D sends two $pull$ data message to peer A and C respectively to retrieve their intermediate results. Finally, D evaluates the query Q_A and sends result to A. The query execution plan is shown in Figure 3, and we depict the whole query processing with the data flow diagram as shown in Figure 4.

5 Evaluation

In order to evaluate the benefits of the dynamic query processing plan, we did some analytical analysis with different settings of acquaintances of peers. The experiments show significant improvements of query execution cost in terms of data transmission which is our primary goal to achieve. We mentioned that our strategy

involves more message passing between peers during execution of a query. However, this does not affect the processing cost severely because the size of each message is very small compared to the shipped data along with the query. In the worst case, we need n extra messages if a query involves n peers.

In Table 4. we show our improvements compared to MQP. In this case, we consider the scenario that we described in Section 5. Table 1 shows the query execution cost as proposed by MQP. Table 2 shows the query execution cost of our proposed strategy. Note that literal values 100, 200, etc. represent the volume the data. For example, entry (200) at column 3 of table 1 shows that volume of data shipped from peer A to B is 200. Therefore, the cost is 200 plus the number of query message, which is 1. We assume that unit time required per data transmission. It is obvious from the two tables that our approach shows significant improvements in terms of data transmission although with some increased number of messages.

6 Conclusion

In this paper, we discussed a dynamic query plan for efficient query processing in P2P environments. The dynamic query plan is generated prior to propagating a query or a part of a query to the participating peers. We mainly focus on minimizing data transmission cost that reduces network traffic. Our proposed methodology shows the improvements over other query processing techniques in terms of query execution cost. We evaluated our technique with a small scale P2P and plan to show performance of the approach with large-scale P2P networks along with efficient routing techniques.

References

[1] MUTE. http://mute-net.sourceforge.net.

[2] JXAT. http://www.jxta.org.

[3] Jabber. http://www.jabber.org.

[4] Napster. http://www.napster.com.

[5] Kazaa. http://www.kazza.com.

[6] Gnutella. http://gnutella.wego.com.

[7] Ratnasamy S., Francis P., Handley M., Karp R., and Shenker S., A Scalable Content Addressable Network. In Proceedings of the ACM SIGCOM Conference, 2001.

[8] Rowstron A. and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Lecture Notes in Computer Science, Vol. 2218, 2001.

[9] Zhao B. Y., Kubiatowicz J. D., and Joseph A. D. Tapestry: An Infrastructure for Fault-tolerant Widearea Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April. 2001.

[10] Kementsietsidis A., Arenas M., and Miller R.J. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In SIGMOD, 2003.

[11] Bernstein P., Giunchiglia F., Kementsietsidis A., and Mylopulos J. Data management for peer-to-peer computing: A vision. In WebDB, 2002.

[12] Rodriguez-Gianolli P., Garzetti M., Jiang L., Kementsietsidis A., Kiringa I., Masud M., Miller R., and Mylopoulos J. Data Sharing in the Hyperion Peer Database System. In VLDB, 2005.

[13] Masud M., Kiringa I., and Kementsietsidis A. Don't Mind Your Vocabulary: Data Sharing Across Heterogeneous Peers. In CoopIS, 2005.

[14] Karnstedt M., Hose K., and Sattler K. Query Routing and Processing in Schema-Based P2P System. In DEXA, 2004.

[15] Papadimos V., Maier D. and Tufte K. Distributed Query Processing and Catalogs for Peer-to-Peer Systems. In CIDR, 2003.

[16] Papadimos V. and Maier D. Mutant Query Plans. In OOPSLA, 2001.

[17] Papadimos V. and Maier D. Distributed Queries without Distributed State. In WebDB, 2002.

[18] Jim T. and Suciu D. Dynamically Distributed Query Evaluation. In ACM PODS Symposium, 2001.

[19] Kossmann D. The State of the Art in Distributed Query Processing. ACM Computing Surveys, Vol. 32, No. 4, pp. 422-469, 2000.

[20] Halevy A. Y., Ives Z. G., Madhavan J., Mork P., Suciu D., and Tatarinov I. The piazza peer-data management system. In IEEE Transactions on Knowledge and Data Engineering, Vol. 16, no. 7, 2004.

[21] Ciglari M. Towards More Effective Message Routing in Unstructured Peer-to-Peer Overlays. IEE Proc. Communications, Vol. 152, No. 5, pp. 673-678, 2005.

[22] Michlmayr E. Ant Algorithms for Search in Unstructured Peer-to-Peer Networks. In ICDEW, 2006

[23] Stoica I., Morris R., Karger D., Kaashoek F., and Balakrishnan H. Chord: Scalable Peer-To-Peer Lookup Service for Internet Applications. In Proceedings of ACM SIGCOMM Conference, pp. 149-160, 2001.