Flexible Networked Data Management on P2P

Pedro Furtado, DEI /CISUC
Universidade de Coimbra, Portugal
pnf@dei.uc.pt

Abstract

One of the interesting research challenges in todays' global repositories is to add more representation and query capacity in a networked P2P-like flexible environment to obtain a flexible networked data management system. With the combination of DBMS expressiveness and P2P flexibility and decentralization, that goal may become a reality. In this paper we describe the techniques we have been working on to combine schema representation and query processing capacity to the flexible and decentralized nature of P2P overlays to obtain a flexible networked DBMS we call the Peer-to-Peer Data Management System (PPDM). Several research issues are raised in the discussion.

1. Introduction

Query processing in P2P overlay networks has been receiving significant attention recently. P2P offers advantages such as flexibility, scalability, decentralization, self-organization, fault tolerance (no single point of failure), while traditional data management systems (DBMS) offer query and schema expressiveness and efficiency. The challenge is to add more representation and query capacity in a networked environment with P2P characteristics, resulting in a peer-to-peer data management system (PPDM) that can be used as a flexible basis for anything from more expressive P2P data sharing to corporate distributed DBMS.

While unstructured P2P systems used message flooding to locate an object, structured P2P systems offer content-based access based on object ID, using *Distributed Hash-Tables* (e.g. CAN [7], Chord [9], Pastry [8], Tapestry [11]). This was a first step towards more powerful query processing functionality. The key issues that need to be solved in order to have a peer-to-peer data management system are:

How can schemas be incorporated and queried in p2p, while abiding to the decentralized and flexible philosophy?

How can we adapt the knowledge we got from complex query processing over parallel and distributed databases into this environment?

How can we automate configuration so that the networked data management system can work and adapt to any environment (LAN, WAN)? The pieces of this puzzle are just starting to be collected presently. Distributed Hash-Tables (DHT) provide a starting point, as they allow efficient and scalable access to objects based on object ID. Efficient use of P2P in more complex schema and query tasks depends on effective techniques to find and retrieve data: simple schemas and lookup queries involve findind objects based on the OID attribute; More elaborate schema and lookup functionality will become available when multi-attribute and range queries can be handled efficiently; Full expressiveness comes with massive processing of complex schemas and queries requiring data shipment between nodes, joins or aggregations and also subqueries. We describe the strategies we have been working on and point out research issues.

Related work on this issue includes MAAN [2,1], a Multi-Attribute Addressable Network which extends Chord to support multi-attribute and range queries and other approaches to querying P2P systems include [4,5]. [5] requires a specialized organization of the P2P overlay into super-peers, which limits its otherwise totally decentralized and flexible nature. MAAN [2, 1] restricts itself to RDF schemas and does not consider the needs of more complex schemas and queries requiring heavy data interchange between nodes. In [4] the authors identified the insufficiency of a very large linear address space (the DHT) to represent complex schemas and queries. We discuss a solution to represent schemas and run not only basic but also complex queries and identify important issues and possible solutions.

2. Basic PPDM Architecture

Figure 1 provides an overview of the PPDM modules in each node. The modules in gray are needed to manage typical P2P functionality. The PPDM requires an additional Data Manager (DM) module to handle schema representation and query processing. The Routing and Location module (RL) is responsible for locating an object and for routing messages over the underlying overlay network architecture. The P2P Storage Manager (SM) is responsible for the administration of object persistence. The node manager (NM) is responsible for adding and removing nodes from the system. Finally, the Data Manager (DM) is responsible for schema definition, indexing and querying. It offers data management

Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05)
1529-4188/05 $20.00 © 2005 **IEEE**

functionality typical of a DBMS but on a different context.

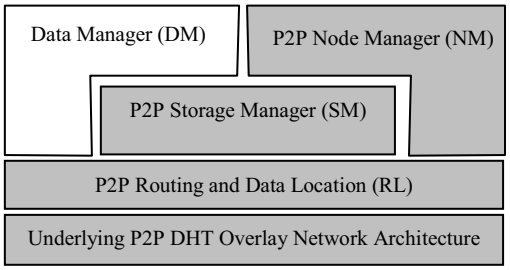

Figure 1 – P2P with Data Manager

3. DHT-based P2P versus DBMS

The PPDM architecture can be applied on any structured P2P overlay (based on DHTs). In this paper we use Chord as an example. The system organizes nodes in an identifier circle modulo 2^m , called the Chord ring. Both the key of a data object and a peer's IP address are hashed into an m-bit identifier. Each node has log N fingers distributed exponentially around the circle and therefore it has to keep only a small amount of state information. A lookup is processed hop-by-hop: the OID of the desired object is hashed to determine the finger leading to the neighbour node that is nearest to the desired object. This ensures that a lookup is processed in at most log N hops and using only log N messages between nodes. This is a totally decentralized and flexible control with no single point of failure, in contrast to parallel and distributed DBMS in which there must be a centralized record and control of all node IPs and therefore also a single point of failure. Although the P2P strategy seems to imply a larger overhead due to the need for hop-by-hop routing, only small messages (e.g. the query) have to be routed this way, as query results are sent directly to the requester node over the underlying IP substrate.

4. Schema Management

We assume a relational schema with tables and tuples over the overlay network, but much of the discussion is also valid for object, XML or other schema representations. For this reason we will use the terms tuple and object interchangeably as well as relation, table or schema object. By default the P2P schema is placed over all the nodes within the overlay network. This is a good strategy for simple lookup queries, but major data shipment requirements of some schemas and workloads may dictate otherwise. A more powerful representation capacity is obtained if we also define primitives for schema space creation, including creation with a specific number of nodes, varied schema and object creation options, security features and flexible entry and exit of nodes from the schema. For instance, in Figure 2 a smaller schema S2 is created over the initial S1 Chord space. The basic Chord ring requires each node to maintain the finger table, pointing at nodes spaced exponentially around the identifier space. Each node maintains the state of *O(log N)* neighbours for a Chord system with N nodes. Non-default schemas are additional Chord rings. Therefore, if a node belongs to the representation of *x* spaces, it must maintain the state of *O(x log N)* neighbours.

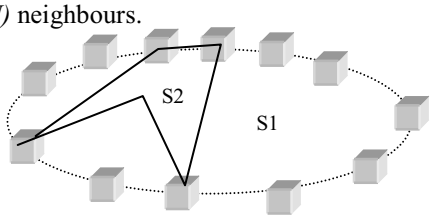

Figure 2: Chord Spaces and Schema Space

Space creation options should be taken into account. These include the option to create schemas exhibiting latency locality or domain locality so that either the schema owner and administrator or an automated workload-based placement optimizer may determine and use the best layout for a given context. Clusters of nearby nodes can be determined using a distance determination strategy such as [6]. Objects can then be created in a schema using alternative placement options, which will be useful for a partitioning and placement algorithm to choose the most appropriate choice: partition using the DHT hash function, replicate or place in a single node:

CREATE TABLE flight(.) in schema S2 partition by fnº;
CREATE TABLE airport(…) in schema S2 replicate;
CREATE TABLE plane(...) on node x;

For instance, small infrequently updated relations can simply be replicated into all nodes to minimize data shipment requirements, while very large relations may be partitioned.

Besides the possibility to define objects as described here, the corresponding metadata and data access information must be stored in some form of catalog and the catalog must also be queriable. The catalog can be similar to a DBMS catalog in a parallel or distributed setting. A simple way to share the catalog with all potential users in a P2P environment would be to also place catalog information in a node whose ID is determined by the hashing of the schema name.

Further research issues related to schemas include security – how user access control to schemas and priviledges would be implemented - and routing of queries into a schema space – how external user queries could be routed efficiently into the schema for processing.

5. Handling Schemas and Queries Efficiently

As we mentioned before in the introduction, the system should be able to handle efficiently simple schemas and lookup queries - find object based on OID attribute; More elaborate schema and lookup functionality – queries specifying multiple attributes or range queries; and the combination of that functionality with massive processing requirements requiring data shipment between nodes and other constructs that require massive data exchange, including joins, aggregations and subqueries.

5.1. Basic Lookup

Basic lookup functionality over object OIDs is already defined in the DHT-based P2P overlay. One attribute a_i is chosen to be the tuple oid, typically the primary key of a relation, and the DHT hashing scheme is used to determine the node into which the tuple is inserted. Consider for instance the relation FLIGHT(*fno, airline, origin, destination, datetime*). If *oid=fno*, tuples can be accessed directly based on *fno* by hashing the *fno* variable to obtain the node ID from which to get the tuple. Object insertion is made via a call to a storage manager (SM) function insert(oid, object), where oid is the object identifier. The SM calls the lookup(oid) function of the RL module, which hashes the oid - HASH(oid) - to determine the node(s) where the object should be placed and stores the object in that node(s). Object retrieval is based on lookup function to locate the node and the retriever gets the object from that node.

5.2. Attribute-Value Location Functionality

In section 5.1. we defined content-based access based on oid. What about content–based access for other attributes? Flooding would be necessary, unless some indexing structure is created instead. We define the Attribute Value Locator (AVL) index for that end, which is an indirection structure accessed via the normal DHT hash value for an attribute value and used to find the nodes where tuples corresponding to that value are located. The AVL is created simply by specifying that an AVL index be created for an attribute (or a set of attributes) of a relation. The AVL is a schema object composed of a set of tuples that are distributed into the schema nodes according to the hash-value for the attribute. The generic sintax for an AVL tuple is:

$AVL[a_i, R_j](v_i, \{nodes\})$

The AVL for an attribute a_i of a relation R_i (or a property a_i of an object R_j) has the tuple(s) of $a_i=v_i$ in node $h(v_i)$, where h() is the DHT hash function. The list {nodes} specifies the nodes where R_j tuples with $a_i=v_i$ are located. For instance, consider attribute Airplane of relation Flight and suppose tuples with the value B747 are available at nodes 1, 6 and 7. The tuple:

AVL[airplane,flight](B747,{1, 6, 7})

Will be available at node with ID hash(B747), for indirection when processing queries including such a condition on the attribute. For instance, in a chord P2P overlay, the following query can be processed using the steps described below:

SELECT * from flight where airplane='B747'

1. The requester node computes hash(B747);
2. The request is sent to node nID = succ(hash(B747)) using the normal Chord routing strategy. The request identifies whether nID should return the node ids to the requester (for further processing before retireving the tuples) or forward the request for the tuples to each of the nodes containing them;
3. The node nID looks in its local database for tuples AVL[airplane,flight](B747,*) and retrieves the node ids;
4. Depending on the option of step 2 the node nID either returns the node ids to the requester (directly over IP) or generates a request for the tuples to each of the nodes;
5. In the second case, the requests are routed into the nodes containing the tuples, which are then returned directly to the initial requester (via IP).

5.3. More Elaborate Lookup Queries

The strategy should also be capable of dealing with multiple attribute queries such as: Find flights tomorrow, from London to Munich. This can be processed with simple AVLs: each condition on individual attributes with available AVLs is used to retrieve node IDs with tuples as described above. Then operations such as disjunction and conjunction are used over the node IDs retrieved from all AVL searches to compute a final list of node IDs with tuples that are then lookuped up.

For additional efficiency, frequently used multiple attribute combinations can be handled faster if it is possible to create a more generic AVL structure that also indexes multiple attributes or expressions. The mAVL structure (multiple attributes) is:

$mAVL[EXPRESSION\ ON\ \{a_i, R_j\}](v_i, \{nodes\})$

This structure is similar to the basic AVL but allows full expressions on combinations of attributes instead of singe attributes.

These strategies should be used by an automated system optimizer and tuner tool that would determine which AVLs should be generated based on the typical query workloads.

Range Queries are another query construct that requires flooding unless some other suitable strategy is devised. In this case the strategy described in MAAN [2] can be applied, which uses an order-preserving hashing to try to access only a subset of the nodes, if possible. This order-preseving hashing will determine a set of node ids that can then be combined with other ones if there are multi-attribute conditions as well in the query.

6. Handling Complex Schemas and Queries

Complex queries are those that may require heavy data exchange between nodes, either data shipment, repartitioning or merging. We have defined a generic query processing strategy for complex queries over a LAN environment in [3]. A similar reasoning applies to complex query processing over the more generic P2P overlay networked environment. Figure 3 shows the steps of the strategy.

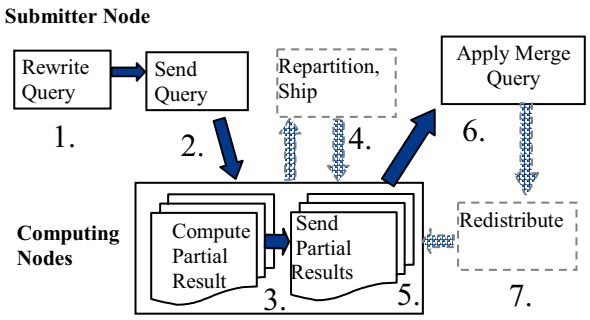

Figure 3 – Complex Query Processing

Queries are submitted and rewritten to determine node queries and merging components. The node queries are sent to the nodes for processing and the partial results are sent back for merging. Major data shipment between nodes or other forms of extra overhead may be necessary during the node computation step (3), sending and merging partial results (5) or redistribution of subquery results. Examples of such overheads include cases in which large amounts of data must be shipped, repartitioning for parallel join, merging results from a large number of nodes into a single one and subqueries requiring multiple processing cycles. These are major potential bottlenecks if schema is dispersed into many, "far away" nodes. In order to prevent this, it is necessary to offer the option of either creation or reorganization of a schema in a smaller schema space exhibiting latency locality, as we proposed in section 4. Additionally, the layout of the relations/objects in the system should be the one that minimizes extra overheads. In [3] we also review workload-based partitioning and placement strategies that can accomplish this objective, including [12].

Finally, it is also important to minimize the overheads associated with the merging of very large amounts of data of partial results coming from processing nodes. This problem can be avoided when processing aggregations. A query such as "select sum(x) from R group by a,b,c" generates a similar query for each node and a merge query suming the partial sums "select sum(partial sums) from INCOMING group by a,b,c". Consider the illustration of Figure 4, where a large number of nodes send their partial aggregations into a single merger node. If there are 1000 nodes, each sending 100MB of data into the merger node, it would need to merge 100GB, computing the final aggregation from 1000 chunks of 100MB each. This problem is avoided using a hierarchial aggregation strategy depicted in Figure 5, whereby groups of x nodes merge their partial results into one successively until the final merging step merges only a similar number of nodes. Using this strategy in the example and x=3 each merger needs to handle only 3x100MB= 300MB.

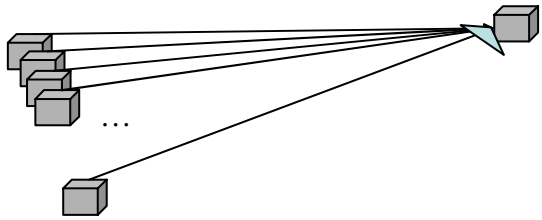

Figure 4 – Aggregation Merging

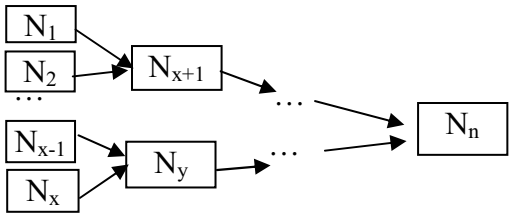

Figure 5 – Hierarchical Aggregation

7. Experimental Testbed

In this section we described shortly the experimental testbed we are using to test the strategies described in this paper. As this is ongoing work and also for lack of space, we only show a preliminary result from the experiments we have been working on using a simulator of a chord P2P overlay. Results shown compare the time taken to answer simple lookup queries with the time taken to answer lookup queries with indirection through AVL.

We devised an inter-network (Figure 6) with two subnet categories, based mainly on link latency/bandwidth considerations: LOCAL - high-speed local network LAN-like with internode latencies of 0.1ms; and GLOBAL – larger latency interconnections

(latencies of 4ms). This is similar to a transit-stub (TS) topology [10].

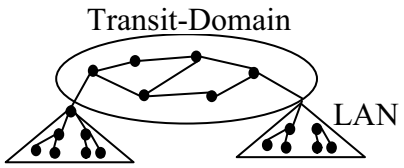

Figure 6: Network Topology

The experimental setup was based on the generation of five 200 node LANs and 40 transit-domain nodes. From the LAN nodes we generated two overlay node sets by picking nodes randomly: a LOCAL set based on picking nodes from a LAN; a GLOBAL set, by picking nodes randomly from all LANs. We tested "overlays" configured with 100 nodes. For these preliminary results we did not consider additional network traffic besides the request. Figure 7 compares the time taken (secs) to answer a lookup query with indirection – when an AVL index had to be accessed before the tuples themselves were retrieved, with the time taken to answer direct oid-based queries. Dashed lines represent local accesses (within a LAN) and filled ones represent global accesses (P2P nodes taken from all LANs).

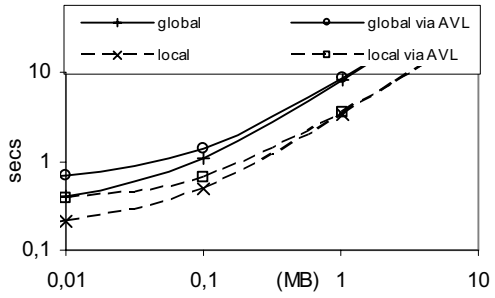

Figure 7 – Lookup Queries with and without Indirection

The extra overhead of the indirection consists of hashing the value to determine the node ID of the required AVL tuples, sending the request to the node containing the AVL tuples, retrieving the node IDs for the answer and sending the request to the answering nodes. In these experiments we do not consider the time taken to access the databases in nodes, as this is dependent on the amount of data. For this experiment the results show that the overhead of the indirection was not large and when the amount of data to be fetched is large the time to retrieve it is much larger. We are currently testing the strategies and considering also all relevant factors to have a thorough comparison of the proposed alternatives.

8. Conclusions and Future Work

In this paper we have argued that it is possible to benefit from both flexibility and decentralization of P2P and expressiveness and efficiency of DBMS. We have sketched how efficient schema management and querying can be implemented on networked DHT-based data management and discussed a lot of open issues that are part of our current work on the subject.

9. References

[1] Cai M., M. Frank, J. Chen, and P. Szekely. A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. In Proceedings of the 13th international conference on World Wide Web, New York, NY, USA.

[2] Cai M., M. Frank, J. Chen, and P. Szekely. MAAN: A multi-attribute addressable network for grid information services. In *4th Int'l Workshop on Grid Computin*g, 2003.

[3] Furtado P.: Efficiently Processing Query-Intensive Databases over a Non-dedicated Local Network. In 19th International Parallel & Distributed Processing Symposium., Denver, USA, April 2005.

[4] Huebsch et al. "Querying the Internet with PIER", in Intl. Conference VLDB 2003: 321-332.

[5] Nejdl, M. Wolpers, et al.. "Super-peer-based routing on peer-to-peer networks. *12th WWW Conferenc*e, May 2003.

[6] Ng, Zang. Predict. Internet Network Distance with Coordinates-Based Approaches. In the 21st IEEE INFOCOMM, Jun 02.

[7] Ratnasamy S. et al. "A scalable content addressable network". In *ACM SIGCOM*M, 2001.

[8] Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Comp.Scienc*e, 2218, 2001.

[9] Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: Scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOM*M, 2001.

[10] Zegura E., K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in Proceedings IEEE Infocom '96, CA, May 1996.

[11] Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry:An infrastructure for fault-tolerant wide-area location and routing. Tech. Report CSD-01-1141, UC Berkeley, 2001.

[12] Zilio, Daniel C., Anant Jhingran, Sriram Padmanabhan, Partitioning Key Selection for a Shared-Nothing Parallel Database System IBM Research Report RC 19820 (87739) 11/10/94 ,T. J. Watson Research Center, Yorktown Heights, NY, October 1994.