A Self-organizing XML P2P Database System

(Extended Abstract)*

Giovanni Conforti, Giorgio Ghelli, Paolo Manghi, and Carlo Sartiani

Dipartimento di Informatica - Università di Pisa - Italy

Abstract. This paper describes XPeer, a *zero-administration* system for sharing and querying XML data. The system allows users to share XML data without significant human intervention, and to pose XQuery FLWR queries against them. XPeer can be used in any application field, being a general purpose XML p2p DBMS, even though its main application is the management of resource descriptions in *GRID* environments.

1 Introduction

The last few years have seen the emerging of the *peer-to-peer* (p2p) computational paradigm, that extends existing ideas about distributed and client-server computing, blurring the distinction between clients and servers. Systems conforming to this paradigm appear as *open-ended* and dynamic networks of peers willing to share computational resources, ranging from CPU cycles to local data, and even to algorithms.

The p2p paradigm was recently adopted in the database community to overcome the limitations of distributed database systems (DDBMS), namely the static topology and the heavy administration work, and to exploit the dissemination of data sources over the Internet.

One key factor in the success of p2p systems, mostly in the field of content sharing, is their easy administration. On the contrary, existing DDBMS require heavy administration efforts, both in the design phase and at run-time: indeed, these systems are based on the presence of global and local schemas, together with their mappings, whose definition and maintenance are a duty of the DBA. Nevertheless, existing p2p systems for XML databases still require significant administration tasks: in Piazza [1], for instance, human intervention is still necessary for defining schema mappings between peers, which implies significant efforts for the DBA, and decreases the dynamicity of the system.

The problem of managing p2p XML databases is quite complex. The source of most issues is the dynamic nature of these systems, where both data and topology may suddenly change. Hence, a closer look at these aspects is necessary.

Changing topology Peer-to-peer systems are usually described as *open-ended* networks of peers willing to share resources. Peers are autonomous, in the sense

* This work was partly funded by the FIRB GRID.IT project.

that they are free to choose the data to contribute to the system, to manage local data without external constraints, and to connect and disconnect at any time. As a consequence, the system is formed by a collection of nodes $\mathcal{S} = \{p_1, \ldots, p_n\}$ that can evolve over time. Topology changes mostly affect the indexing structures used for routing queries. For instance, if a node p_i containing data (let's say a set of XML nodes s) relevant for a query q suddenly becomes unreachable, then any index entry associating p_i to s should be updated to avoid unnecessary messages, or, in the worst case, *run-time* problems.

Local Updates Peer autonomy implies that peers have the right to update their data, even if shared, at any time. In particular, peers can perform both *value* and *schema* changing updates (unlike in relational databases, the loose structure of XML data blurs the distinction between value and schema updates). Value and schema updates influence query mediation and query routing since sudden data changes may invalidate existing query plans or routing structures, hence imposing potentially expensive updates of distributed index structures. Moreover, most *schema-driven* data management approaches (see [1]) are severely affected by local updates, hence requiring human intervention for adapting the system to the new data.

Our Contribution This paper describes a *zero-administration* p2p system for sharing and querying XML data (XPeer). The system allows users to share XML data and to pose XQuery FLWR queries against them without any significant human intervention (the user still has to write her own queries). The system, based on a hybrid p2p architecture, *self-organizes* its superpeer network, and allows for arbitrary changes in the network topology.

2 XPeer Overview

Basics XPeer is an XML p2p database system, which manages data dispersed over an *open-ended* network of autonomous peers. In XPeer no constraints are imposed over exported data, that can be freely updated by their owner; moreover, nodes can join and leave the system at any time, so the system has a dynamic topology. Exported data are integrated in a *blind* way, i.e., no *global schema* is defined: this solution allows for a significant decrease in the administration load of the system. Of course, this fundamental choice restricts the applicability of the approach to situations where schema mapping can be avoided, or can be performed out of the p2p system (i.e., by a local schema adapter). We believe the choice is perfectly reasonable in the application field we are targeting first (resource description).

XPeer adopts a hybrid p2p architecture [2], where peer nodes may also perform administrative tasks (acting both as peers and as *superpeers*). Databases hosted by XPeer can be queried with a proper subset of XQuery. Due to the complexity of the system, and, in particular, to the changing topology of the system, no guarantee about the completeness of query results can be provided.

Data Model Data in the system are represented as unordered forests of node-labeled trees. Each tree is augmented with the indication of the hosting peer (*location* in the following) as well as with a freshness parameter fr, which indicates when the last update on the tree was performed ($\perp$ indicates that the freshness is undefined, and it is necessary to ensure that the model is closed). To support freshness parameters, the data model has a universal constant τ, which denotes the current global time in the system: since query results are assumed to be incomplete, the assumption of the existence of a global time is feasible.

Query Language The query language of choice is the FLWR subset of XQuery [3] without universally quantified predicates and sorting operations. The choice of the FLWR core of XQuery distinguishes XPeer from most existing p2p systems, which are limited to simple *key-lookup* queries, or to linear path queries, and which require significant modifications to support *full* database queries [4].

3 XPeer Architecture

XPeer is a *hybrid* p2p system composed by a dynamic set $\mathcal{S} = \{p_1, \ldots, p_n\}$ of autonomous peers, which share data and execute global queries on the database. Some nodes in $\mathcal{S}$ (in most cases, those with adequate computational power and/or network bandwidth) perform administration tasks too: these nodes, called *superpeers*, form a set $\mathcal{SP} \subseteq \mathcal{S}$. Peers become superpeers on a voluntary basis, and retain their peer role. We favor a hybrid p2p architecture wrt a hierarchical one (e.g., the GRID GRIS/GIIS system) since it offers more robustness to failures and it can adapt more easily to network changes.

Peer Network Peers share XML data and execute queries on top of these data. Peers export a description of the data being shared in the form of a tree-shaped DataGuide [5], called *tree-guide*, which is automatically inferred from the data by means of a tree search algorithm. Leaf nodes in the schema are endowed with statistical information about value ranges, to allow the system to better identify relevant data sources during query compilation. The following Example shows a sample XML document and its tree-guide.

Example 1. Consider the following document, hosted by a peer p_1, describing buildings in a real-estate market database.

```
<market>
   <buildings>
      <building> <desc> Very nice flat in the Upper East Side </desc>
                 <location> Upper East Side, Manhattan </location>
                 <price> 1350000 </price>
                 <type> comdo </type> </building>
      ...
      <building> <desc> Elegant luxury house in the countryside </desc>
                 <location> Greensboro </location>
                 <price> 1700000 </price> </building>
   </buildings>
</market>
```

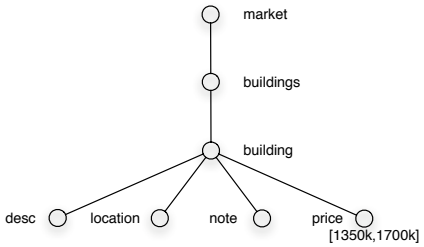


Fig. 1. A sample tree-guide.

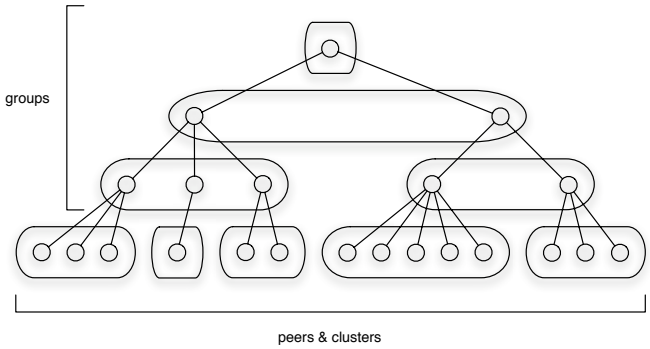
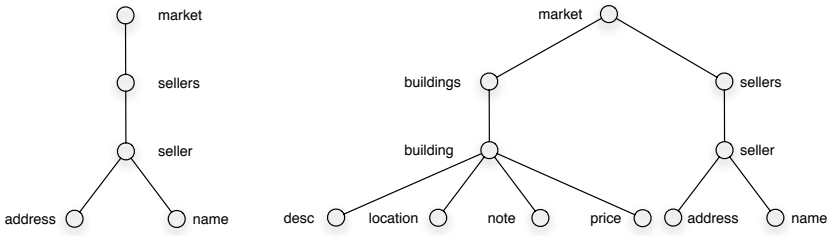


Fig. 2. Overall logical system architecture.

The corresponding tree-guide contains each distinct path in the document, endowed with statistical information about value ranges (e.g., the range 1350000−1700000 for price elements), as shown in Figure 1.

Clusters Peers are logically organized into clusters of nodes, where each cluster contains one superpeer, which is in charge with the management of the cluster, e.g., the compilation of user queries and the management of peer information. Peer clustering allows the system to decrease the efforts required for compiling queries. To this aim, clusters are formed, whenever it is possible, on a *schema-similarity* basis, i.e., peers exporting data with similar schemas are clustered together (the system still works even if nodes in the same cluster have very different schemas). Inside any cluster, some peer may (partially or totally) replicate the content of other peers in the cluster. Replicas are built to balance the workload in the cluster and to exploit peers with huge computational resources, and are valid up to a given time. The replication process, as many other processes in XPeer, happens on a voluntary basis.

SuperPeer Network Superpeers have the duties of tracking topology changes, managing schema information, and compiling user queries. Superpeers are organized to form a tree, where each node hosts schema information about its



(a) Another sample tree-guide.

(b) A superpeer schema.

Fig. 3. Another tree-guide and a super-peer schema.

children; superpeers having the same father form a *group* (which is very close to a peer cluster). The resulting logical topology is shown in Figure 2. Superpeers host two kinds of schema information about their children: the list of the schemas of their children (the *schema list*); and the union of these schemas (the *superpeer schema*). The schema list is used during query compilation for identifying relevant data sources, or superpeers whose descendants can contain relevant data; the superpeer schema, instead, is passed to the father as schema of the superpeer, and it is built without any schema integration activity, so that no human assistance is required.

The following Example shows a sample superpeer schema.

Example 2. Consider the following XML document, hosted by a peer p_2, describing seller information in the real-estate market.

```
<market>
   <sellers>
      <seller> <name> Patrick Bateman </name>
               <address> 25, Park Avenue </address>
               <phone> ... </phone> </seller>
      <seller>  <name> Tim Price </name> </seller>
   </sellers>
</market>
```

This document can be represented by the tree-guide shown in Figure 3(a). Assuming that both peers p_1 (see Example 1) and p_2 have the same superpeer *sp*, then the superpeer schema of *sp* is depicted in Figure 3(b).

Network Evolution The topology of the network can evolve over time. To adapt the organization of the superpeer hierarchy to changes in the network, superpeers may split clusters and groups, and may ask for new superpeers. In particular, when the workload for a given superpeer *sp* becomes too hard, *sp* first tries to relocate some of its children in other clusters/groups (*network balancing*); if the problem persists, *sp* then asks the system for new superpeers, and delegates them

part of its workload (*network extension*); if the workload is still too heavy, *sp* can finally disconnect some of its children (*peer de-gnoming*). On the other hand, when the workload for a given superpeer *sp* becomes too light, *sp* may decide to import some children from busy superpeers, or it may decide to relocate its children to another superpeer, and then to exit the superpeer network (*network contraction*).

4 XPeer Query Processing

XPeer supports the FLWR core of XQuery, the standard query language for XML data being developed by W3C [3]. Since data are usually dispersed among many peers, XPeer does not preserve the document order in query results.

FLWR queries are translated into algebraic expressions, and are executed on the system by relying on *data-integration-like* techniques. To speed up query execution and to decrease peer and superpeer workload, the system exploits mechanisms for replicating peer content, and for caching query plans and query results; these mechanisms can be ignored on an explicit request by the user.

Query Algebra The query algebra of XPeer, further described in [6], is an evolution of the query algebra for centralized XML data described in [7]. The query algebra consists of three classes of operators. The first class contains operators that navigate unordered forests of node-labeled trees, binding nodes to variables, and that build new XML trees from existing variables bindings (*path* and *return*); the second class, instead, contains operators for manipulating tuples of variable bindings, as in standard OO query algebras [8] (σ, π, $\bowtie$, $DJoin$, etc); the third class, finally, is formed by operators for managing *locations* (the algebraic counterpart of peers), and, in particular, for uniting their content ($LocUnion$) and for inserting replication constraints into query plans ($Choice$). Since location choices are guarded by temporal parameters, the query algebra data model has been enriched with the universal constant τ, describing the system global time, and with time labels for locations.

Query Compilation Query compilation is performed in two phases. In the first step, a query is submitted by the user to a peer p_i. p_i translates the query into a triple $Q = (q, \tau', \delta_{\tau'})$, where q is a *location-free* algebraic expression, i.e., an algebraic expression with "holes" (called *spots*) in place of locations, τ' is the query submission time, and $\delta_{\tau'}$ is a user-defined freshness parameter; in particular, $\delta_{\tau'}$ indicates that the system may use replicas and caches synced after time $\tau' - \delta_{\tau'}$, and allows the user to specify freshness and quality requirements for the result of the query (e.g., $\delta_{\tau'} = 0$ means that only up-to-date caches and replicas can be used, while $\delta_{\tau'} = \infty$ means that any existing cache or replica can be used). In the second phase, p_i sends the query Q to the superpeer network, via the superpeer of its own cluster, for the compilation of a *location assignment* ρ, i.e., a function assigning unions ($\bullet$) and choices ($|$) of locations to location spots. This compilation is performed in a hierarchical way by matching the twigs of q with schema

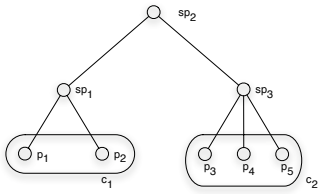


Fig. 4. Another system topology.

information, and by traversing the superpeer hierarchy till any interesting location has been detected. In particular, the superpeer responsible for the cluster of p_i matches the twigs of the query with the schemas of its children peers, hence finding all relevant locations in the cluster; then, the superpeer sends the query to the super-peer responsible for its group, which in turn matches the query twigs against the schemas of its children, and resends the query to clusters that may contain relevant data. The query is also propagated up in the hierarchy to find all relevant locations. The query compilation process, hence, requires the system to propagate the query till the root of the super-peer network, but still limits the exploration of the network to a fraction of the hierarchy. Once the location assignment ρ is computed, ρ is passed to the issuing peer p_i for query execution; by making p_i responsible for the execution of its query, the system minimizes the load of the superpeer network.

Query Execution Once the issuing peer p_i has received the location assignment ρ for a query Q, it applies common algebraic rewriting to the fully specified algebraic expression, such as selection push-down and distribution of unions, and then starts executing the query, which is split into *single-location* sub-queries that are sent to the corresponding peers; p_i waits for query results, and then executes operations, such as joins, involving data coming from multiple sources. Query subexpressions are locally optimized and executed by system peers, hence allowing each peer to choose the best execution strategy for any given algebraic expression. Query decomposition is performed by exploiting an algorithm close to that of YAT [9]: the algorithm just browses the algebraic tree in the search of maximal single-location subexpressions, which correspond to peer sub-queries.

Example 3. Consider the following XQuery query:

```
for $b in input()//building,
    $d in $b/desc,
    $p in $b/price
return <entry> {$d, $p} </entry>
```

This query returns the description and the price of each building in the *real-estate* market database. Assume that the system has the structure shown in Figure 4, where p_1 and p_2 contain the documents described in Examples 1 and 2 respectively, while p_3, p_4, and p_5 contain data about loans and mortgages. Suppose the user submits the query to p_2, p_2 builds the following *location-free* algebraic expression and then sends it to sp_1.

$$return_{entry[\nu\$d,\nu\$p]}(path_{(//,\$b,in)building[(/,\$d,in)desc[\emptyset],(/,\$p,in)price[\emptyset]]}(\mathbf{spot_1}))$$

sp_1 matches the query twig against the list of tree-guides of its peers, hence finding p_1 relevant for the query, and then propagates the query to sp_2; sp_2, in turn, matches the query twig over its schema list, hence excluding the descendants of sp_3 from the query plan. As a consequence, $\mathbf{spot_1}$ is replaced by loc_1.

5 Conclusions

This paper describes the architecture of XPeer, a p2p XML data management system. The architecture of the system is *self-organizing*, in that the superpeer network can adapt its structure to changes in the system network topology and in the query workload. Furthermore, the system requires no human intervention for its administration, hence being a *zero-administration* DBMS.

XPeer is a general purpose XML p2p database system, so it can be used in any application field. Still, its main application is the management of resource descriptions in a *GRID-like* environment: in particular, XPeer should form the basic infrastructure for extending (and, eventually, replacing) the *LDAP-based* resource discovery layer of existing GRID systems.

References

1. Halevy, A.Y., Ives, Z.G., Mork, P., Tatarinov, I.: Piazza: data management infrastructure for semantic web applications. In WWW2003, Budapest, Hungary, 2003
2. Yang, B., Garcia-Molina, H.: Designing a Super-peer Network. In ICDE 2003, Bangalore, India, 5-8 March 2003, IEEE Computer Society (2003)
3. Boag, S., Chamberlin, D., Fernandez, M.F., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML Query Language. Technical report, World Wide Web Consortium (2003) W3C Working Draft.
4. Harren, M., Hellerstein, J. M., Huebsch, R., Thau Loo, B., Shenker, S., Stoica, I.: Complex Queries in DHT-based Peer-to-Peer Networks. In: IPTPS 2002, pages 242-259
5. Goldman, R., Widom, J.: DataGuides: Enabling query formulation and optimization in semistructured databases. In: VLDB'97, pages 436–445
6. Sartiani, C.: A Query Algebra for XML P2P Databases (2003) Manuscript draft. Available at `http://www.di.unipi.it/~sartiani/papers/eve.pdf`.
7. Sartiani, C., Albano, A.: Yet Another Query Algebra For XML Data. In IDEAS 2002, Edmonton, Canada, July 17-19, 2002. (2002)
8. Sophie Cluet and Guido Moerkotte. Classification and optimization of nested queries in object bases. Technical report, University of Karlsruhe, 1994.
9. Siméon, J.: Intégration de sources de données hétérogènes. PhD thesis, Université Paris XI (1999)
10. Papadimos, V., Maier, D., Tufte, K.: Distributed Query Processing and Catalogs for Peer-to-Peer Systems. In: CIDR 2003, Asilomar, CA, USA, January 5-8, 2003. (2003)
11. Sartiani, C., Ghelli, G., Manghi, P., Conforti, G.: Xpeer: A self-organizing XML P2P Database System. In Proceedings of P2P&DB 2004.