Routing XQuery in A P2P Network Using Adaptable Trie-Indexes

Florin Dragan, Georges Gardarin, Laurent Yeh
PRISM laboratory
Versailles University & Oxymel, France
{firstname.lastname@prism.uvsq.fr}

Abstract

The increasing popularity of XML and P2P networks has generated much interest in distributed processing of XML data. In this paper, we propose a novel solution for schema and content indexing in a P2P architecture. Our solution is based on a hierarchy of super-peers managing progressively compressed summaries of XML schemas. The main features of the extended tries used for schema and XML data indexing are scalability, as the index adapts to the different memory capacities of peers, and efficiency as it is very simple to provide routing decisions based on path-set matching. We summarize the results of experiments that prove the validity of the proposed distributed trie-indexes. This work is partly included in the Satine European Project that develops a P2P semantic mediator for publishing and discovering web services.

Keywords

P2P, XML, mediation, routing, index

1. INTRODUCTION

In recent years, P2P systems have been extensively developed (Androutsellis-Theotokis 2004). Mostly used in file sharing applications, they provide a flexible medium for accessing large amounts of data. Among the main qualities of P2P systems we point out: fault tolerance, self-reorganization, adaptation, reliability, scalability. These properties show the maturity of P2P systems and should contribute to the introduction of the P2P paradigm in the context of data management systems.

One important issue in P2P systems is how to localize peers that contain data relevant for a certain query. A system that implements a good policy of routing queries for reaching all the peers with relevant data tends to be very efficient at retrieval time and result completeness.

In a pure networking approach (no DHTs), the XML routing algorithm has to take into consideration the structure of the XML data. The query forwarding process has to be based on a complete path-matching algorithm (routing to nodes that satisfy a set of paths). In other words, every peer taking routing decisions has to maintain an index with the XML document paths of its neighbors, and shall forward the query according to the matching between the query paths and the document paths.

In the last years, several solutions have been proposed for name-based routing in P2P networks (Androutsellis-Theotokis 2004). However there is not much work for XML routing based on predefined hierarchical structures (Papadimos 2003) and on XML schemas (more precisely, the tree structure of documents called treeguide) repositories (Sartiani 2003). XPeer (Sartiani 2003) is a "self-organizing XML P2P database system" that facilitates storing and querying of XML data. The authors use a hierarchical organization of SPs. However, a SP stores all the schema of his children. This means that the index at SPs from higher levels is not scalable with the number of peers. In (Papadimos 2003), a P2P architecture is presented as suitable for distributed XML querying based on a global data namespace. Data retrieval is made using distributed catalogs. The catalog "servers" represent centralization points that may introduce serious bottlenecks and that are susceptible to exterior attacks. MediaPeer (the system referred by this paper) is a hybrid P2P data sharing system. It introduces an efficient and compact data structure (the trie-index) used for routing XQueries. It does not suffer from heavy indexes or somehow centralized catalogs to discover data localization.

In this paper, we present a structural index adapted to the different peer resources of a P2P system and a routing technique based on the proposed index. Our contributions are:

- **A scalable indexing structure** used for referencing paths in an XML document.
- **A structural routing method** for XML path-set queries based on the trie-indexes.

In our approach, similarly to (Dogac 2004) or (Oser 2003), the overlay network is composed of peers and Super-Peers (**SP**). Peers are clustered to SPs according to their metadata information and SPs may be in turn clustered to other **SPs**. Document fragments are retrieved using the XQuery language. Every XQuery is translated to a collection of path-sets routed by each SP to peers that contain relevant data. The forwarding decisions are taken based on the index structures maintained at each SP in the overlay network. For taking correct forward decisions, each SP has to index the treeguides published by its clustered peers (or SPs). For reducing the size of the index at each level and for taking fast routing decisions, each SP index is organized as an adaptable trie (Morrison 1968) with a more reduced size for the SPs at upper levels.

The rest of the paper is organized as follows. In section 2, we give a brief presentation of MediaPeer (Dragan 2005), the P2P data mediation system based on distributed trie-indexes. The trie-indexes and their interesting properties are presented in section 3. In section 4, we outline the query processing algorithm through an example. In section 5, we summarize some experiences proving the characteristics of trie-indexes. Finally, we conclude in section 6.

2. MEDIAPEER ARCHITECTURE

MediaPeer is a P2P data mediation system built on a hierarchical network formed of peers and SPs. Every peer that takes part in the network may share XML data by providing an XML abstracted view of the data it is willing to share. This abstraction is a set of tree-like structures, one for each collection of queryable XML documents. It is called a treeguide (Sartiani 2004).

According to its resources (memory and/or computational) and to its network availability, any peer may also play one or multiple SP roles. A peer becomes a SP when nominated by other peers or SPs. SPs are chosen among a list of computers known as sufficiently stable in the network. A SP allocates a fixed amount of main memory to routing tables, which also gives a limit for routing overhead as the routing computing time is logarithmic in number of entries in the routing table. SPs manage structural and value indexes. At nomination time, a SP receives treeguides, possibly summarized, from his children peers or SPs. According to the available memory, it keeps in the main memory the whole or a compressed treeguide, as explained below. A SP also deals with query routing. It receives requests for data localization and based on the index analysis it determines other peers or SPs to pass the requests to. More details can be found in (Dragan 2005).

3. TRIE-INDEXES FOR ROUTING

In this section, we describe the indexing data structure and the XQuery routing process based on hierarchical summaries of DTDs or schemas, managed under the form of compact tries.

3.1 Trie-index organization

Each node in the overlay network must provide a routing function. A SP cannot in general store in the main memory all the treeguides of all the sources it handles. Another requirement of the routing process is that it must be done very fast. In other words, a SP cannot search an XML path-set in all treeguides: it would be too costly in CPU time.

For that, the routing process requires the existence of a distributed index structure that must contain summaries of the relevant paths of the documents, i.e., parts of the treeguides. We have adapted a structure model that is used in IP-routing, for dealing with XML path-sets. We summarize the treeguides as tries

(Morrison 1968) for being able to index more XML paths. Furthermore, we index only the truncated part of each element name of the paths so that the trie-index fits in the main memory of the corresponding SP. The fast prefix matching capability of tries and the variable compact storing model that characterizes these structures make them well adapted to routing path-based queries. Thence, a trie-index is customized to the memory available on each hosting SP.

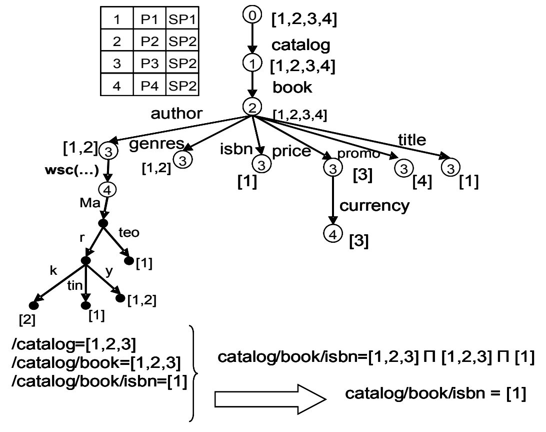

Figure 1. A trie at SP level

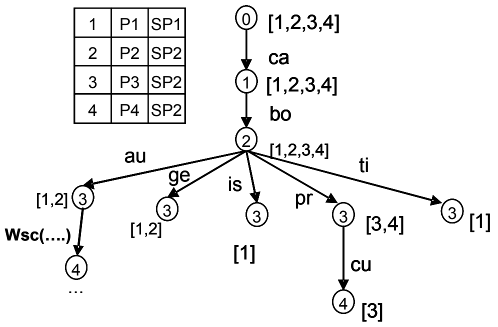

Figure 2. A compressed trie-index

In a trie that indexes the treeguide of an XML document, the vertices represent partial (or entire) strings of element names and the nodes give information about the position of the element names in the paths. As presented in Figure 1, the adaptation of a trie also consists in adding to every node the identifiers of the peers that contain the path (starting from the root). For compactness and ease of maintenance (e.g., removing a peer), a peer is referenced in the index by a simple ID (in brackets in Figure 1). We use an integer for representing the ID of a peer and the translation to the real IP address is made using a separate redirection table. Associated to the trie, we also maintain a table giving the references of the children peers or SPs.

For values indexing, when submitting its schema description, a peer may declare a value summary Web Service (possibly only for certain elements in the schema). When the schema is inserted in the trie-index, the Web Service is called and the returned values are added to the target of the corresponding paths. The values are trie-indexed as strings (based on the alphabetical order). In figure 1, it was declared a WebService that returns the possible values for the "author" element. The peers that published the data respond to the web-service calls. Periodically or on administrator demand, calling the Web Service to update the values summary simply refreshes the index. This has some similarities with active documents in KadoP (Abiteboul 2004).

3.2 The routing function

The routing function determines the nodes to address a path localization request. It is based on a search for a set of strings summarizing the path in the trie-index. For finding the ID of the peer that contains data according to a certain path we use the algorithm `FindID` sketched in figure 3.

Based on figure 1, we give an example of the execution of the algorithm on the given trie-index. The path `/catalog/book/isbn` is decomposed in `/catalog`, `/catalog/book` and `/catalog/book/isbn`. The list of nodes that contain /catalog is [1,2,3], that contain /catalog/book is [1,2,3], and that contain /catalog/book/isbn is [1]. The intersection is [1], meaning that only the node with ID=P1 may contain XML data according to the path `/catalog/book/isbn and` that the next hop in the routing process is SP1.

3.3 Trie-index size reduction

As introduced above, every SP maintains an index as a trie, which describes the XML paths published by the SPs (or peers) at lower levels. The trie-index decreases in precision from level to level starting from the root of the overlay network. For reaching scalability, the size of the routing data is reduced at upper levels. Thus, the trie at the top of the hierarchy remains small with a low precision but covers all its dependent nodes.

Every trie becomes lighter from one level to another by simply reducing the number of characters in the representation of the element names (path-units). A SP contains a trie-index that represents the merging of all the trie-indexes of his children. In the case that a SP has not enough memory, the trie-index is reduced according to the algorithm `ReduceTrie` also sketched in figure 3.

```
FindID(Path)
  Decompose the Path in path-units.
  Compute sub-paths of Path from path-units.
  For each sub-path do
   Find the list with the IDs of the nodes
that contain the current sub-path.
  Return the intersection of the lists
computed at the previous step.
```

```
ReduceTrie(Trie, MSize)
  While (Size (Trie) > MSize)
    For each path-unit ∈ Trie
      If CanReduce(path-unit)
        Reduce_Size(path-unit)
  Factorize path-units with
common prefix
```

Figure 3. The algorithms used for finding the forwarding ID and for reducing a trie-index

The actual function of reduction depends on the desired final precision (for example, we may reduce the size with only one character per path-unit or we may get rid of half of the characters in the representation). According to the desired selectivity, we can reduce the size of only certain path units. In this case, some sub-tries will be more precise. Figure 2 presents the structural part of the trie from figure 1 after several steps in the reduction process. We can further eliminate all nodes with no ramification, which are useless for selection, as done in Patricia trie (Morrison 1968).

4. QUERY EXECUTION

In this section, we present how the queries are processed in a network following the hierarchical overlay organization of MediaPeer, where SPs use trie-indexes for indexing data paths. An XQuery is decomposed in a collection of path-sets that must be satisfied. Alternative paths can also be handled, but we consider mandatory paths for simplicity. A query issued at a peer is introduced as a set of path-sets into the SP network and routed to peers with relevant data. An example is illustrated by the figure 4.

For $a in collection("CATALOG")/catalog /book for $b in collection("REVIEWS")/review where $a/author contains("Mary") and $b/book/isbn = $a/isbn and $b/p contains "P2P" return {$a/price}	→ extract two path-sets	<PathSet> <path>/catalog/book/author["Mary"]</path> <path>/catalog/book/isbn </path> <path>/catalog/book/price </path> </PathSet> <PathSet> <path>/review/book/isbn </path> <path>/review/p["P2P"]</path> </PathSet>

Figure 4. Extraction of two path-sets

Each path-set is forwarded to the index root SP. Every SP that receives a path-set compares each path in the set with the paths in the local trie-index and finds the ID of the next-hop children. When value web-services are declared a value matching method is applied. Thus, the requests are passed down in the overlay tree. When a request reaches a peer, the peer checks for the real existence of the paths in its published schema and returns the relevant data directly to the requesting client. Several algorithms are currently compared to effectively run the query under partial results of sources.

5. SOME EXPERIENCES

For size reduction, we prove the adaptation of trie-indexes to different memory requirements, by measuring the evolution of the size of trie-indexes during different reduction phases for 5 sets of paths. Although it is obvious that a trie will reduce its size, we cannot predict the reduction factor because of the compact path indexing. The results of the measurements are presented in figure 5.

As it may be observed, by reducing the length of the element name representation, the size of the trie indexes reduces with ~30%. By applying a Patricia trie compression technique (removal off all the nodes with a single outgoing vertex), we reduce the size with a ratio around 50%.

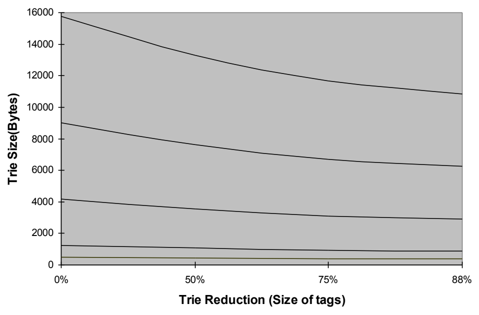

Figure 5. Size reduction of a trie-index by deleting characters in tag (path-unit) encoding

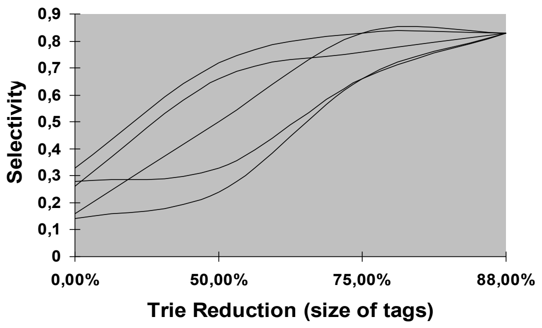

Figure 6. Variation of trie selectivity: the horizontal axis represents the reduction percentage of the original tags

When we eliminate characters from tag representation, the precision of trie-indexes reduces. This means that a SP will forward the incoming queries to more of his children. In the worst case all the queries will be down-flooded to all the children. Thence, it is important to see how the same query becomes more selective when the size of the trie-index reduces by the decreasing the number of characters in tag representation (figure 6). A bigger selectivity means that more paths in a path-set return positive results when matched against the trie. However more we descend in the SP hierarchy, more the filters becomes efficient reducing the false routes.

6. CONCLUSION

In this article, we have introduced an efficient technique for indexing XML schemas in a hierarchical overlay network and a technique for structurally routing XQueries . The main features of the presented index are: scalability (as the index has an adaptable size) and efficiency (as the algorithm of matching a path against a trie-index has a reduced complexity). We believe that using path-sets and trie-indexes is a good basis for discovering relevant data sources in a Web-enabled XQuery mediator. Progressively compacted trie-indexes provide a nice method for abstracting things. At the upper levels are represented only compact abstracts and at the lower levels are represented details. More experiences will be reported in future articles.

REFERENCES

Abiteboul, S. et al, 2004. KadoP: Knowledge and Data in Peer to Peer. http://www-rocq.inria.fr/gemo/KadoP/

Androutsellis-Theotokis, S. et al, 2004. A survey of peer-to-peer content distribution technologies. *In Computer Surveys,* Vol. 36, No. 4, pp. 335-371.

Dogac A. et al, 2004. The Satine Project: A P2P Network for Discovering Web Services. http://www.srdc.metu.edu.tr/webpage/projects/satine/

Dragan, F. et al, 2005. MediaPeer: a Safe, Scalable P2P Architecture for XML Query Processing. *Globe'05 workshop* Copenhagen. Denmark.

Morrison, D. R. 1968. PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric. *Journal of the Association of Computing Machinary*. Vol. 15, No. 4, pp. 514-534.

Oser, A. L. et al, 2003. Information integration in schema-based peer-to-peer networks. *Proceedings of the 15th Conference On Advanced Information Systems Engineering (CAISE 03).* Klagenfurt/Velden, Austria.

Papadimos, V. et al, 2003. Distributed Query Processing and Catalogs for Peer-to-Peer Systems. Conference on Innovative Data Systems Research (CIDR). Asilomar, USA.

Sartiani, C. et al, 2003. XPeer: A Self-organizing XML P2P Database System. *Proceedings of the First EDBT Workshop on P2P and Databases (P2P&DB 2004)*. Crete, Greece.