MediaPeer: a Safe, Scalable P2P Architecture for XML Query Processing

Florin Dragan, Georges Gardarin, Laurent Yeh
PRISM Laboratory, Versailles University, FRANCE
{firstname.lastname@prism.uvsq.fr}

Abstract

Increasing popularity of XML and P2P networks has generated much interest in distributed processing of XML data. We propose a novel solution organized around a mediator capable of processing XQueries over multiple heterogeneous data sources. The solution consists in a hierarchical overlay network formed of peers and super-peers indexing the XML path of data sources. Our solution is: (i) Scalable as the network can grow dynamically with size adaptative trie-based indexes in each super-peer. (ii) Reliable as procedures are developed for recovering from node failures or root saturations. (iii) Efficient as query processing is done by an existing optimized mediator that can track query process progresses and response sizes.

1. Introduction

In recent years, P2P systems have been extensively developed. 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, and scalability. These properties 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 peer organization and query routing tends to be very efficient at retrieval time and result completeness.

In our system named MediaPeer, the overlay network is composed of peers and super-peers, where peers are clustered to super-peers according to their metadata information and super-peers are also clustered to other super-peers. Every XQuery is translated to a union of path-sets (i.e., the XML paths referenced in the query) and the forwarding decisions are taken based on the index structures maintained at each super-peer in the overlay network. Routing indices extending techniques described in [5] but based on trie indexes are used. The indexing technique and the proper query execution process are presented in [1].

The organization of super-peers in a tree-like structure is a well-known solution (it was also proposed in other architectures, see for example [2]). We choose it for its capabilities to reduce the path search space to about O(logn) and to integrate several algorithms regarding the scalability, dynamicity and fault tolerance. We present these algorithms in this paper.

Our architecture is easily deployable because a super-peer (a peer that will participate in the routing process) represents a process running at the machine containing a peer. This means that a systems that shares data in the peer-to-peer network may also participate in the routing process; the strategy of choosing super-peers is made according to the physical properties and availability of the peers.

The rest of the paper is organized as follows. In section 2, we give a complete overview of the overlay network structure and develop the peer, super-peer architectures. In section 3, we present some solutions for dealing with the dynamic behaviour of the network. Section 4 is devoted to network recovery after failure of a node. We report on some experiments in section 5, and we conclude in section 6.

2. The architecture of MediaPeer

MediaPeer is a P2P data mediation system built on a hierarchical network formed of peers and super-peers. In this section, we describe the topology of the network and the architectures of a peer and a super-peer.

2.1 Network Tree Topology

MediaPeer is an overlay peer-to-peer network architecture that facilitates sharing and retrieval of XML data. Physically the network is formed only of peers but logically it assumes the existence of peers and super-peers for efficiently clustering and routing the data. Any peer may be a client, when requesting data, or server when answering to other peer queries. The super-peers serve only at routing the data localization demands to peers that share relevant data. Peers are clustered to super-peers and super-peers are clustered to super-peers based on data-description criteria. The overlay network has a tree-like topology with peers disposed as leaves and super-peers disposed on different intermediate hierarchical levels.

2.2 Overview of a Peer

Every peer that takes part in the network may share XML data by providing an XML Schema abstraction of the local data. This abstraction is a set of XML tree structures, one for each collection of queryable XML

documents. In MediaPeer we use a simple abstraction index – a trie index [1].

Our architecture is based on an existing mediator XLive [3]. The XLive mediator is a data integration middleware managing XML views of heterogeneous data sources. Using XLive mediator one can integrate heterogeneous data sources without replicating their data while the sources remain autonomous. Basically, the P2P overlay network is used to localize the data processed by the mediator distributed engine.

The architecture of a peer node is represented in figure 1. It encapsulates one or more data sources in XML/XQuery. The mediator through specific wrappers can query several data sources of different types. We experiment with relational DBMSs (Oracle, MySQL) but also with XML native DBMSs (Xyleme, X-Hive)[4]. The mediator provides three services, one to publish path-sets of XML views of the sources, another to accept queries, and finally one to publish results in XML. In our current system, we assume for simplicity that all XML schemas are derived from a unique ontology and can be integrated through XQuery without mapping problems. We shall introduce semantics mappings expressed in XQuery between XML schemas in a future version, similarly to [6] for example.

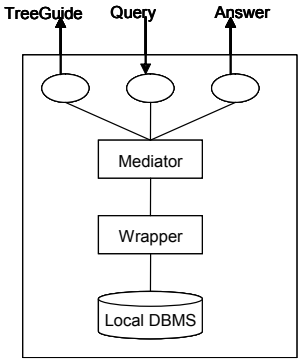

Figure 1. Architecture of a peer

2.3 Overview of a Super-peer

According to its resources (memory or computational) and to its network availability, any peer may also play one or multiples super-peer roles. This means that if a peer has been connected into the network for a long time and has enough memory to store an index, it may become a super-peer and, besides asking and answering queries, it also helps in routing queries.

The reason that motivated our choice is that in a computer network (i.e., the Internet) and especially in a peer-to-peer network the only entities that commonly take part in the communication and processing are computers with similar resources and availability. Thence, in our network, on every node may reside one or multiple super-peer processes and always one peer process. Considering that peers have a dynamical behaviour (frequently leave or enter the network), the super-peers that help in routing the queries may be not stable, which may cause organisational problems. These problems are treated in the following sections.

3. Network Dynamic Evolution

In this section, we analyse the dynamic behaviour of the network under normal conditions (i.e., no computer or software failure), including load balancing. The overlay network structure evolves in the following cases: (1) New peer insertion. (2) Peer deletion (on a voluntarily basis). (3) Peer data source structural changes. (4) Super-peer insertion. (5) Super-peer deletion.

MediaPeer implements evolution strategies inspired from the B-tree reorganisation algorithms. As with practical implementations of B-trees, we fix a maximum degree D for the tree and a maximum height H. D and H are parameters dependent on the network size. A node of degree greater than D is split in two. If it is the root, it is also split, except in the situation when the tree has reached height H. In this case, we implement a special policy to deal with network saturation that will be explained in the next section. Contrary to other architectures, only the super-peers reorganise according to the variable network load. We have chosen this approach because reorganisations are performed at low cost.

3.1 Adding a Peer

When a new peer wants to join the network, the following actions are performed: (i) Localisation of a super-peer to cluster the new peer. (ii) Update of all the super-peer trie-indexes, from the peer to the root.

For being able to enter the network, a new peer must know the address of a peer or of a super-peer. Several techniques for finding the suitable network entry point are possible, but a simple one is to use a server that maintains a reference to an entry point. Then, the peer that wants to enter the network sends to the entry point an entry-demand with his IP address.

The system has to determine the best cluster to enter the peer in. For that, the demand is routed into the network until it reaches the root super-peer. The demand evaluation is at first made in a root-coordinated manner. The root super-peer sends a path-set request with his IP address to the initiator of the demand, which replies with its local path-set (given by the mediator metadata service).

Next, the root forwards the path-set insertion demand to its children into the network for finding the suitable super-peer to connect to, i.e., the best cluster. A super-peer receiving the path-set insertion demand computes a similarity measure with its trie-index. The similarity measure quantifies the differences between the super-peer index-trie and the new path-set. Sophisticated measures can be used, but we start with a simple one giving the number of paths appearing in the new path-set but not in the super-peer index. Measures are sent back to the parent, which determines the best fit. In the case where several super-peers return the same maximum similarity, the child super-peer that provides the most available resources is selected.

The selected super-peer then forwards the demand to its children. The process goes on until the demand reaches the lowest level in the hierarchy, which determines the super-peer that cluster the new peer. All selected super-

peers on the descending path also update their trie-index adding information extracted from the path-set concerning the new peer data. A confirmation of insertion is forwarded up the tree, so that all concerned peers commit their trie-index updates including the new peer. Before updating the local trie-index, every super-peer may perform different reduction strategies upon the incoming path-units [1].

The implemented strategy takes advantage of the hierarchical organization of the network. When connecting a new node, only the most similar super-peers have to process the new peer metadata. In this case the cost of inserting a new peer is O(h*k), where h is the height of the network tree and k is the average number of children. It is a top-down strategy that benefits from the peer clustering for fast insertion of a new peer.

3.2 Deleting a Peer

The procedure of deleting a peer is initiated on demand from the user, when she/he wants to leave the network. The peer sends a disconnection demand to his cluster super-peer. The demand has to go up to the root passing by all the super-peers in the hierarchy (DOWN-UP). The disconnection message must contain information regarding the identification of the peer so that all the peers on the ascending path will modify accordingly their trie-indexes. The peer identifier is simply searched in the trie, and all corresponding entries are removed. Paths with no peer associated are also removed.

3.3 Updating a Peer

An update upon an XML data source structure (DTD or schema) at a peer changing some paths requires propagation in the super-peer network to maintain the trie-indexes. We proceed then by deleting the peer and reinserting it in the network. We plan to optimize path-sets changes in the future. Versions numbers could be used for managing trie-index updates in real time.

3.4 Adding a Super-Peer

A super-peer is added to the network to keep it efficient, when a resource manager of a super-peer detects an overloading. An overloading arises when one of the following conditions becomes true: (i) The number of children of a super-peer exceeds the maximum degree D of the network (similar to the order of a B-tree). (ii) The resource manager detects an overload in message number, or memory size (trie-index too large with no reduction possibility), or average request processing time. The overlay-network structure changes are triggered only by stable conditions (i.e, the conditions are met during a time period). The addition of a super-peer may lead to increase the number of clusters containing peers with similar data.

To distribute the load, the redistribution of the children (super-peers or peers) of the overloaded super-peer to other super-peers has to be performed. This shall be done under stress conditions for the overloaded super-peer. To avoid resource contention, we choose to create two new super-peers, prepare them, and finally integrate them in the network in place of the overloaded super-peer. The advantage of this strategy is to avoid disturbing the overloaded super-peer during the adding process.

Let SP0 be the saturated super-peer. The splitting process is composed of the following steps:

1. SP0 initiates a demand for a new SuperPeer (SPn).
2. SP0 sends to SPn the local trie-index.
3. SPn initiates a demand for a new SuperPeer (SPm).
4. The trie-index and children are distributed between SPn and SPm.
5. SPn and SPm open connections to SP0 father and SP0 children.
6. The old connections are closed.
7. SP0 leaves the network.

It is very important to mention that during the splitting process the network is running. The proposed algorithm guarantees a continuous network.

The process of load distribution may be recursively propagated up to the root of the hierarchy if the parent node detects an overloading condition. It is important to mention that a node undertakes a split decision only after a period of overloading to avoid cascading reorganisation due to node splitting. When the process reaches the root node the network hierarchy shall increase with another level.

3.5 Deleting a Super-Peer

Deletion is the situation opposed to that presented in the previous section. It may generate the regrouping of super-peers from the same level. The regrouping is valuable because it reduces the number of super-peers and, consequently, shortens the number of nodes that a source localisation request has to follow.

Our deletion algorithm works as follows. When a super-peer detects a reduced load, it demands to its children a report with the status of their activity. Following a cost function threshold, the super-peer may decide the fusion of two children. The cost function should integrate the message load of the children, the sizes of their trie-index, and the resource capacity of the host systems.

To fusion two super-peers, our algorithm first creates a new super-peer to perform the fusion. This allows the children to continue their routing role during fusion. The fusion is controlled by the parent, which demands the creation of a new super-peer (SPn). SPn then asks the merging children for their trie-index and merges them. When this process is finished, an index update is required to the parent super-peer for replacing the merged children by SPn in the trie-index. After the update is finished, the two children super-peers are destroyed as they are replaced by (SPn) by the quick update of the parent trie-index (children identifiers removed and replaced). It is necessary that the parent super-peer blocks all the re-organisational processes while the replacement procedure takes place.

3.6 Choosing a Super-Peer Host

When adding or deleting a super-peer, we create new super-peers (as explained above) that have to be hosted somewhere. The problem addressed in this sub-section is how to choose the computer that hosts a new super-peer.

Generally, a computer that maintains a new super peer has to respect some constraints, among them: (i) It must not already host some "neighbour" super-peers. For enhancing the network reliability, a new super-peer should not be hosted on the computer where already resides one of its brothers, parents, or children. (ii) It must provide the resources required for the new super-peer. For solving the first constraint, the super-peer that initiates the creation demand send a list with the super-peers that must not be selected. For the second constraint, the resource manager checks if enough local resources exist to satisfy the demand.

The nomination process proceeds as follows. A super-peer initiates a new super-peer demand. It is sent to certain favourite addresses from the local list of contacts. If a super-peer receiving a demand is not able to meet the necessary resource requirements, it forwards the request to the addresses from his list of contacts. On the contrary, if the local computer has enough resources and is able to host a new super-peer, it holds the required resources for a time period waiting for a resource locking demand. After the time period has expired, if no locking demand is arrived, the resources are released. Thus, the demand for new resources is forwarded in an organized fashion starting from most favourite super-peers and ending with the less favourite ones.

4. Network Reliability and Recovery

In this section, we describe the efficient recovery procedures proposed for the MediaPeer network.

4.1 Super-Peer Failure

This case appears when a super-peer (SP) leaves the network due to an internal failure. There are two possible situations: (i) A node (peer or super-peer) detects the failure of its parent when passing a request up. (ii) A SP detects the failure of one of its child when sending a request down.

In the first situation we adopt a level collaboration solution. This means that all the super-peers from the same level will collaborate in order to restore the functionality of the network. When a node is not able to send a message to his father SP, it contacts the sibling nodes (nodes on the same level in the hierarchy with same father) and a demand is issued for the creation of a new father SP process (for the children) (to synchronize the node negotiation for finding a new super-peer, the control is given to the node with the smallest IP address). The node responsible for the creation of a new SP follows the algorithm presented in the previous section. The new created SP receives the addresses of all its children and directly starts the treeguide index generation by merging and reducing the children trie-indexes.

For the second case we adopt a bottom-up technique. The children will collaborate for rebuilding a functional network. For that when a SP finds that one of its children does not respond, it sends a message requesting to one of the children of the failed SP to generate up a null request. Next, if the failure is still detected by a non-answer, the reorganization process takes place as described in the previous paragraph. In summary, the children nodes (peers or super-peers) always initiate the recovery procedure for a super-peer, as they are the ones knowing (partially) the path-sets to synthesize at the father. Figure 3 illustrates the super-peer recovery procedure.

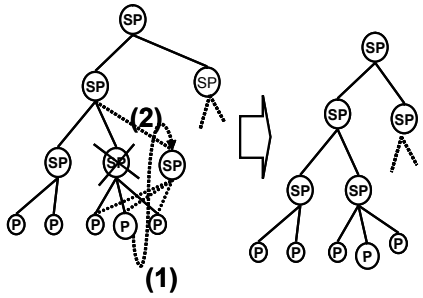

Figure 3. Super-peer recovery procedure

4.2 Peer Failure

A peer failure is normally detected by a super-peer that sends messages to the peer and does not receive any acknoledgment. In this case, the super-peer should answer the localisation requests, even if the peer is known as not answering. Two solutions are possible: the pessimistic one supposes that the super-peer does not answer to the different requests involving that peer; the optimistic one assumes the super-peer always answer positively. The first one will reduce overhead in query processing and seems to be preferable. In other words, it declares the source as non-reachable.

4.3 Urgent Peer Elimination / Disconnection

A computer that hosts a peer and one or more super-peers, when leaving the peer-to-peer network, requires the deletion of the peer and the replacement of the super-peers. In this case, a demand is sent directly to the "Resource Manager" of one super-peer that sends disconnection messages to all other super-peers on the same system. The local super-peers make demands for new super-peers generation (to replace them in the overlay network) and accordingly, transfer them all the administrative information.

4.4 Root Saturation

There are two situations that generate root saturation:

- The root memory becomes insufficient for indexing all the peer path-sets (index-saturation). The index-saturation at hierarchy root is solved by performing trie-index reductions by indexing by prefix of labels, as described in [1].
- There is an overload of messages into the network that leads to an outsized routing load at the root of the hierarchy (load-saturation), while the maximal height of the network has already been reached.

For the last case, to avoid the overloading of the root peer with routing messages, we propose a technique of level flooding. This means that in the situation when a super-peer forwarding up a routing message discovers that his father is overloaded, it shall transmit the message to all its sibling peers. In this way, all the messages are forwarded

to peers with relevant indexes (no relevant peer will be skipped). In the same time the load at super-peer will be maintained constant. By using level flooding we take advantage of the flooding features (fast propagation) for a certain level. Depending on the branching "size" per level, this process may speed up the query routing process. In figure 4, we give an example of level flooding.

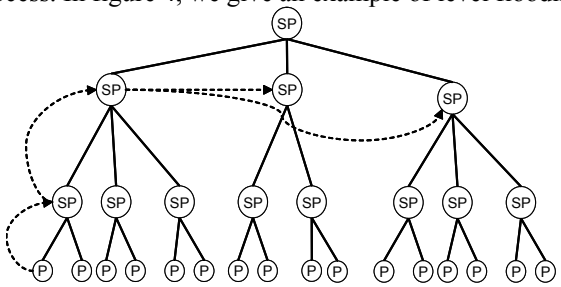

Figure 4. Level Flooding

We present in the next section the results of simulations made over our overlay-network model, that show the reduction of traffic at the root super-peer in the presence of level-flooding.

5. Some Experiences

5.1 Hierarchical network routing

We have simulated the query routing process over a particular model of hierarchical network (every super-peer clusters two other super-peers and four peers) with a variable number of peers (8,16,32). For being able to find the different query paths, we consider that each query is addressed to only one peer. 8, 16, 32 queries were simulated. The path followed by each one was analysed by counting the number of super-peers that participate in the routing process and the number of links crossed by a query. By adding different costs to every query action (routing, network transferring, index search, etc.), a realistic cost of routing in our model of hierarchical network is obtained.

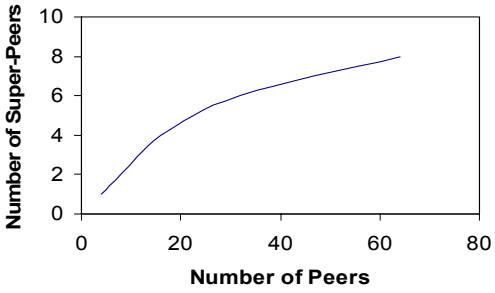

Figure 5. Number of Routing Super-Peers

According to Figure 5, we observe that the number of super-peers that take part in the routing process does not increase proportionally with the number of peers (due to the proposed clustering). This means that it is more advantageous to route queries using a hierarchical network than to use "flooding". In our architecture, the effective number of computers that communicate for performing the routing may be even more reduced (considering that two super-peers may be localized on the same physical machine).

Observation: We have analysed our most favorable case when a query is not duplicated at the super-peer level to be sent over multiple paths. However, considering that the peers are clustered according to their schemas, it is expected that the number of super-peers that are used for routing in general cases be close to our values.

5.2 Level flooding

For evaluating the effect of "level flooding,", we have measured the load at the root of the overlay network. The two curves in figure 6 show the load at the root in the presence (down)/ absence (up) of level flooding.

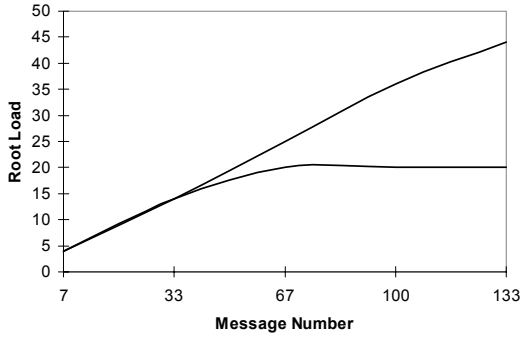

Figure 6. The influence of level-flooding

As presented in the graphic, the level flooding method is suitable for distributing the load from a higher level to lower level super-peers. By using level-flooding, the system becomes more scalable and maintains the search-space dividing features of a tree.

6. Conclusion

In this article, we have presented an easy deployable P2P infrastructure whose purpose is to integrate large collections of data distributed over large communities of peers. The main features of the presented architecture are: fast data localisation based on tree-organized super-peers, fault tolerance achieved with the network self-repairing architecture, easy deployment as to share data a peer must install some lite modules that will perform the network tasks. MediaPeer relies on general solutions for scalability, dynamicity, and fault tolerance in a P2P network.

Future work includes more measurements of query performance under stressing conditions for the network. Currently, the localisation demands include simple filtering operators; more can be achieved with for example joins, where the network could be used for efficient join processing. Further work is also required to evaluate partial result policies.

Furthermore, as mentioned above, we plan to include a semantic level for translating local path-sets in global terms according to some global ontologies. Mapping path-sets to global ontologies is already operational in the Satine project [7] we are working on.

7. References

[1] Florin Dragan, Georges Gardarin, Laurent Yeh "Routing XQuery in a P2P Network using Adaptable Trie Indexes" Technical Report, PRISM Laboratory, FRANCE, submitted for publication.

[2] Carlo Sartiani, Paolo Manghi, Giorgio Ghelli, and Giovanni Conforti. XPeer: A Self-organizing XML P2P Database System. In Proceedings of the First EDBT Workshop on P2P and Databases (P2P&DB 2004), Crete, Greece, 2003

[3] Dang-Ngoc, 2003. Tuyet-Tram Dang-Ngoc, Georges Gardarin.: Federating Heterogeneous Data Sources With XML, IASTED IKS 2003: Scottsdale, AZ, USA, Nov. 2003.

[4] Florin Dragan, Georges Gardarin "Benchmarking an XML Mediator", to appear, ICEIS 2005, Miami, May 2005.

[5] Crespo, A., Garcia-Molina, H. Routing indices for peer-to-peer systems. In Proceedings International Conference on Distributed Computing Systems (July 2002).

[6] Halevy, A. Y., Ives, Z. G., Mork, P., Tatarinov, I. Piazza: Data management infrastructure for semantic web applications. In Proceedings of the Twelfth International World Wide Web Conference (WWW2003) (Budapest, Hungary, May 2003).

[7] Dogac, A., Y. Kabak, G. Laleci, S. Sinir, A. Yildiz, A. Tumer, " SATINE Project : Exploiting Web Services in the Travel Industry ", eChallenges 2004 (e-2004), 27 - 29 October 2004, Vienna, Austria.