XRPC: efficient distributed query processing on heterogeneous XQuery engines
Zhang, Y.

Citation for published version (APA):
Zhang, Y. (2010). XRPC: efficient distributed query processing on heterogeneous XQuery engines

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
P2P content sharing applications have gained enormous popularity in less than a decade. But nowadays, the demand of more sophisticated P2P applications that go beyond simple keywords search is growing. However, developing P2P applications that provide non-trivial distributed data management facilities is a rather cumbersome task, since applications have to deal with information from huge collections of highly heterogeneous and volatile data sources. A P2P data management system which acts as a database middle-ware and offers a uniform database abstraction on top of a dynamic set of distributed data sources, should ease the development of data-intensive P2P applications. In this PhD work, we research which features such a database abstraction should offer and how it can be realised efficiently by extending and combining existing XDBMS with P2P technologies.

8.1 Research Summary

The main research question we try to answer in this thesis is:

*How to support efficient processing of full-fledged XQuery queries – including those containing XQUF expressions – on large amounts of XML data served by heterogeneous XQuery engines in P2P settings?*

The main research question is divided into five more specific questions, as stated in Chapter 1. Below we summarise the contributions of this thesis as the answers to those specific research questions.

1. How should we extend XQuery with a query shipping mechanism that is suitable for the targeted environments?

Before extending XQuery, we have first carefully analysed the characteristics of our target environments and defined a list of criteria which such extensions must satisfy (Section 3.1). The result is XRPC, a simple but powerful extension of XQuery that adds the RPC paradigm to XQuery. At the syntax level, the “execute at” statement of XRPC is inspired by that of XQueryD [144, 29]. However, XRPC has several properties that makes it particularly suitable for large scale P2P settings.

XRPC is simple because *(i)* it adds RPC in the least invasive way to XQuery: adding a destination URI to an XQuery function application; *(ii)* it respects well-accepted (de factor) Web standards, e.g., RPC, SOAP, HTTP and Java; *(iii)* the SOAP XRPC protocol is a stateless protocol; and *(iv)* it is easy for other XQuery engines to adopt XRPC (we
will come back to this point in the next research question). As an orthogonal extension, XRPC preserves the expressiveness of XQuery, and the SOAP XRPC protocol supports all XDM data types, including user-defined XML Schema types with the ability to validate the SOAP messages.

XRPC is flexible because (i) an XRPC call can be made anywhere in a query where an XQuery function call is allowed; (ii) modules only need to be imported once and all the imported functions can be executed both locally and remotely on an arbitrary number of peers, avoiding any additionally effort to compile a module.

XRPC is efficient because (i) the Bulk RPC technique not only greatly reduces network latency, but also exploits the set-at-a-time infrastructure of DBMSs, e.g., turning bulk selections into a join between the SOAP message and the XML documents; (ii) with the by-projection decomposition algorithm, almost every XQuery expression can be decomposed, giving a large number of possibilities for distributed query optimisation; and (iii) the runtime projection algorithm can be much more accurate than the compile time techniques, which minimises the sizes of the SOAP messages. The efficiency of XRPC also implies that it is scalable with respect to the number of peers in the network (achieving minimal number of network round-trips thanks to the stateless protocol), the number of documents and the sizes of the documents.

However, XRPC is not only a language extension. The SOAP XRPC network protocol makes XRPC the only distributed XQuery proposal that is interoperable and full-fledged, at the time of this writing\(^1\). Within the scope of the XRPC project, we have also studied topics such as distributed transaction management and query optimisation. These topics are addressed by the remaining four research questions.

### 2. How can different XQuery engines be united to jointly evaluate a single query?

This problem calls for an interoperable and easy to support solution. Concerning interoperability, the basic building blocks of XRPC, i.e., XML, XQuery, SOAP and HTTP, are all well-defined and generally accepted Web standards. To enable communication among different XQuery engines, we propose a well-defined SOAP-based protocol, the SOAP XRPC protocol, in Section 3.3. Thus, network communication in XRPC uses XML messages over HTTP. Such a protocol is easy for existing XQuery engines to support, since they are already perfectly equipped to process XML messages, and there is ubiquitous support for URIs and HTTP.

The design of XRPC has been kept as simple as possible, which makes it easy to understand and support. RPC is an obvious and popular paradigm for implementing the client-server model of distributed computing\(^2\). Hence, to support XRPC, an XQuery engine merely needs to extend its grammar rules to support the “execute at” statement and implement the stub code and request handler. Serialisation and parsing of the SOAP messages are functionalities that already exist in every XQuery engine.

---

\(^1\) XButler [134] is the only distributed XQuery proposal that also adopts an open communication protocol, i.e., SOAP RPC. However, due to the limitations in the supported data types by SOAP RPC (as discussed in Chapter 1), XButler is restricted to support functions, which only have atomic values as their parameters and results, like the WSDL services.

\(^2\) The idea of RPC was first described in RFC707 [145] in 1976.
An XQuery engine can even participate in the evaluation of a distributed XQuery query without having XRPC integrated. The XRPC Wrapper, described in Section 4.2, is a SOAP service handler, implemented in Java and XQuery for reasons of interoperability. It can be run on top of any XQuery systems that is XQuery 1.0 compliant. The XRPC Wrapper translates a SOAP XRPC request message into a standard XQuery query, which is passed to the underlying XQuery engine for execution. Query results are wrapped in an XRPC response envelope and sent back to the caller. Extracting information from a request message (i.e., information of the called function and its parameter values) is done using XPath steps and generating the XRPC response message is done using element construction, again, features that are available in every XQuery engine.

3. How are distributed updating queries supported?

Since the XRPC extension is orthogonal to all XQuery features, it automatically supports remote execution of XQUF expressions by means of calling updating functions on remote peers. If updates are allowed in a system, dealing with transactional semantics is a matter of course. In this Ph.D. work, we have spent some effort on formally specifying the semantics of distributed updating XRPC queries and the support for atomically committing distributed transactions. In Section 3.4, we have formally defined the semantics for both read-only and updating XRPC queries under different levels of isolation, and the necessary extensions to the SOAP XRPC message format. To support atomic committing of distributed transactions, we have chosen not to extend the SOAP XRPC protocol with any 2PC-like features, but rather to rely on the recent industry standard Web Services Atomic Transaction [55, 54], which provides a SOAP-based 2PC interface. Section 4.5 describes how updating queries are handled utilising this standard. By using the XRPC Wrapper, we have demonstrated processing of distributed updating XRPC queries involving Galax, MonetDB/XQuery, Saxon and X-Hive [181].

If multiple groups of XML nodes are inserted by multiple uses of the same kind of insert expression with the same target node, XQUF leaves the ordering among these groups to be implementation-dependent, which implies an undeterministic ordering. However, node order plays an important role in XML and XQuery. We thus regard it worthwhile to define a deterministic update order that simply respect the order in which updates appear in for-loops and sequence constructions. In XRPC, a deterministic distributed updates order can be ensured by a simple extension to the SOAP XRPC protocol, as described in Section 4.4. Finally, in Section 5.7.2 we extend the semantics of XQUF to allow updates on remote documents which are identified by an xrpc:// URI scheme. In this section, we also propose rewriting rules to ensure the correctness of decomposed queries that contain updates on remote peers, that is, updating expressions may only be carried out on the peer owning the document to be updated.

4. How can we automatically decompose XQuery queries for distributed execution?

Decomposing queries to address multiple data sources is a well-known optimisation mechanism in distributed query processing. While many of the existing techniques can be carried over in distributed XQuery processing, there are challenges, introduced by the XML data model and the XQuery language, which do not exist in a value-based relational data

---

3This includes insert into, or insert as first|last into, or insert before|after.
model. That is, XML nodes have node identities and structural properties. In Chapter 5, we studied in detail automatic decomposition of standard XQuery queries with semantic guarantees. We have presented three decomposition algorithms for both read-only and updating queries. All three algorithms use a copy-based protocol (i.e., the SOAP XRPC protocol and its extensions) for (un)marshalling function parameters and results. In Chapter 1, we have argued our choice for a stateless protocol, which we believe is more suitable for our target large scale P2P environments than a stateful protocol. In Chapter 5, we show, especially with the by-projection algorithm, that such a simple stateless protocol is extremely efficient and gives sufficient freedom for distributed query optimisation. The correctness of all decomposition algorithms are formally proven in Chapter 6, which adds a strong theoretic foundation to the algorithms.

5. How can we integrate existing DBMS with P2P overlay networks to provide non-trivial data management facilities to P2P applications?

During this Ph.D. work, we have taken some preliminary steps toward an integration of existing XDBMS and DHT network structures. This is described in Section 4.6. The basic idea is to avoid any additional extension to the XQuery language. Section 4.6 proposes two different ways of coupling an XDBMS with one or more DHT networks, i.e., loose coupling and tight coupling. Both couplings rely on a new dht:// scheme to address resources in the underlying DHT network. In the tight coupling, we have to extend the DHT API with a new function so that a DHT peer can benefit from its local XDBMS to enable a more complex querying facility than the standard DHT API, with which only a complete document can be stored and retrieved by its name.

The XRPC remote function execution mechanism and the ideas of MonetDB/XQuery⋆ are applied in a P2P application called StreetTiVo [182]. StreetTiVo enables near real-time search in video contents by distributed and parallel execution of compute-intensive video analysis tasks on multiple peers. Our work on the StreetTiVo application confirms our assumption that a P2P middle-ware DBMS could ease the development of data-intensive P2P applications. With XRPC, the rather complex functionalities of StreetTiVo were quickly implemented using just a handful XQuery functions, which in turn are executed on the participating machines.

Theories proposed in this Ph.D. work have also been adopted in practice. XRPC has been included in the open-source XDBMS MonetDB/XQuery, which can be downloaded via http://monetdb.cwi.nl/Download/. The software includes the XRPC client and server for distributed XQuery processing, a Java package containing an XRPC Wrapper for cross-system XQuery processing, and an interface for directly making XRPC calls from within web pages. User experience has shown that XRPC is an easy to use mechanism for distributed XML querying.

8.2 Future Work

With this Ph.D. work, we have taken the first steps towards building a middle-ware PDBMS. However, there are still plenty of open areas for future research work. The following questions are particularly interesting.

Distributed Query Placement With the by-projection decomposition algorithm, we are able to decompose almost all XQuery expressions. A natural next step is to decide the placement
for each decomposed subexpression, i.e., to find the optimal peers on which a subexpression should be executed. Basically, this needs a cost-based optimisation that takes network, CPU and data distribution into account [68, 21, 80, 187, 188]. However, in P2P setting, it is unrealistic to assume the availability of these statistics. As a possible solution direction, one could contemplate using runtime methods to improve optimisation quality. One idea is to express peer selection criteria in XQuery expressions and attach such expressions to the destination URI of each \texttt{execute at} statement. At runtime, when the query (with the peer selection rules added) is evaluated, it will select the best remote peers at that moment, for each decomposed subquery to be executed. We could also borrow ideas from the execution model of Mutant Query Plans [136, 137], in which distributed query plans are evaluated incrementally (by trying to resolve as many as possible logical URNs into URLs at each peer). However, we will not consider exchanging query plans among peers for reasons of interoperability.

**Scalable P2P Transaction Management** So far, we have only used a strict 2PC protocol for transaction management. In P2P settings, we deem it important to define less strict but more scalable protocols to manage transactions. Distributed Snapshot Isolation (DSI) is especially interesting because it requires weaker locking protocols, which makes it likely to perform better than a classical two-phase locking protocol in high-latency WAN environments. To our knowledge, there has not been much previous work on Distributed Snapshot Isolation, while lately in commercial applications (centralised) snapshot isolation has found wide user acceptance. One idea here is to use Lamport Clocks [116] as the timestamp for DSI, providing the notion of Lamport consistency. The objectives of such work would contain a formal definition of this consistency criterion, as well as an analysis of the protocols needed. The StreetTiVo application could be used to validate this approach.

**Query Optimisation in MonetDB/XQuery** Our initial work described in MonetDB/XQuery needs more work to become mature. A first goal of using DHTs is to create “logical URL’s that specify XML data items without necessarily pinning down the actual URL (hostname, path). Such logical URLs may even be used as synonyms for XML data items that are spread over multiple locations. Another benefit of using DHTs is to allow \(O(\log(N))\) network cost equi-selections (\(N\) is the number of peers) and add self-managing properties to the distributed system under churn, i.e. maintaining connectivity, and providing an automatic replication mechanism that prevents data loss. Apart from these design aspects of the XDBMS-DHT integration, another interesting topic is to research which query optimisation possibilities the proposed tight DHT coupling can provide.