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.
Introduction

1.1 Motivation

The use of the Internet as the means for individuals and organizations to interact and exchange information has significant consequences for data management technologies. The data management research and industry communities have therefore embraced the W3C recommendations around XML, which is the de facto standard for information exchange on the Internet\(^1\). Around XML an ecosystem of other Web standards has emerged, including XML Schema, SOAP, WSDL, XSLT, XPath and XQuery. While the design of these Web standards is in principle not a question of computer science research, rather one of design by committee, the fact that the Internet has adopted these standards as the means of (semi-)structured data exchange creates a reality that provides relevance and urgency for finding answers to new research questions, such as presented here in this thesis.

Generally speaking, one can view XML data on the Internet as a huge wide-area XML database containing semi-structured information. The individual machines on the Internet belong to truly billions of different individuals and millions of different organizations, and consequently act as independent peers. The term “peer” is used here, as in principle there is no hierarchy among these machines. Cooperative software systems where multiple peers work together without necessarily a central point of control, are called Peer-to-Peer (P2P) systems. Also, building systems that combine data from multiple peers on the fly is a new way to create rich applications with little effort (“mashups”). As such, the question is how the creation of such cooperative peer systems can be facilitated.

To ease the development of data-intensive P2P applications, we envision a P2P Database Management System (P2P DBMS) that acts as a database middle-ware system. It manages dynamic collections of heterogeneous data sources (peers, with different software installed) and provides a uniform database abstraction to the application. The goal of this Ph.D. work at large is to research which features such a database abstraction should offer, and how it can be realized efficiently by extending and combining existing Database Management Systems with P2P technologies.

Please note here, that P2P DBMS means something different than the common notion of P2P systems, which the general public knows best as P2P file sharing systems, often used for illegal downloading of copyrighted media. Whereas a P2P file sharing system typically manages (binary) files, a P2P DBMS manages (semi-)structured information. In the context

\(^1\)We could not resist using this by now infamous phrase.
of this study we focus on data in XML format, i.e., on P2P XML Database Management Systems (P2P XDBMS). File sharing systems typically only allow simple queries, based on meta data of the files, e.g. using keywords or simple properties. P2P XML database systems, in contrast, allow users to query the contents and structures of the XML data, using an XML query language. In the context of this thesis, we assume the query language to be XQuery (which is a super-set of XPath).

In our quest for creating P2P XDBMS technology, we first focused on Distributed XDBMS technology. The distinction between Distributed and P2P technology is that in the former, users (e.g., application programmers) are aware of on which sites (i.e., peers) data are located. Distributed queries typically involve specific and explicit locations where data is to be queried from. In P2P systems that also target large environments, where users cannot keep track which data is on which peer, and where the group membership is highly volatile (peers enter and leave continuously and unpredictably), users are typically shielded from explicit knowledge where data is located. We focused first on Distributed XDBMS technology as this area also was unexplored, and Distributed XDBMS technology can be seen as a building block for P2P XDBMS technology.

In this research work, we have hence focused on distributed XML DBMS aspects including query execution, query optimisation, and transaction management. The result of this work is XRPC (stands for Xquery RPC), a minimal XQuery extension that enables efficient distributed querying of heterogeneous XQuery data sources. In the remainder of this section, we motivate the choices we made in the design and implementation of XRPC, and in how XRPC is used for distributed XQuery processing over heterogeneous data sources.

1.1.1 Interoperability

The primary design goal of XRPC is to create a distributed query mechanism with which different query processors at different sites can jointly execute queries. An important way for a system to achieve interoperability with other systems is to adhere to published interface standards. For this reason, our choice for the basic building bricks of XRPC naturally falls on XML, XQuery, SOAP and HTTP – all well-defined and generally accepted Web standards.

After its first public revealing at the SGML 1990 Conference, XML has quickly gained enormous popularity as a data exchange format among different applications and organizations. XML owns its success to several properties: (i) it is hardware and (to some extent) software independent; (ii) it is a self-documenting format, i.e., a single document contains both description of the structure and field names, and values; (iii) it is suitable for data with or without a clear structure, thus one can capture plain text, and text with some structure, e.g., e-mails, all the way to very structured information such as tuples; and (iv) XML schemas are extensible, which makes it easy to support backward compatibility. Consequently, the choice for XML as the data model and XQuery as the query language – and web standards in general – eases many aspects of distributed data management.

The only way for two different systems to communicate is to use an open protocol. However, to the best of our knowledge, none of the existing proposals of distributed XML query processing use or define such an open standard protocol. Some of them use a non-open protocol, e.g., AXML [9], and some of them use a proprietary protocol, e.g., DXQ [70]. In such systems, heterogeneity is hard to achieve, or impossible. Therefore, we choose to use an open network protocol to enable communication among different XQuery engines. Although W3C has a Candidate Recommendation XML Fragment Interchange [87] and SOAP has a SOAP
CHAPTER 1. INTRODUCTION

RPC subprotocol, these two standards have a common problem: the data types they support do not match the data types defined by the XQuery Data Model (XDM) [71]. The XML Fragment Interchange defines a way to send fragments of an XML document – regardless of whether the fragments are predetermined entities or not – without having to send all of the containing document up to the fragments in question. Thus, XML Fragment Interchange only supports XML elements, but no atomic values; while SOAP RPC only supports exchanging of atomic values without XML elements. Neither approach deals with (XQuery) sequences of heterogeneous types.

As a result, XRPC also encompasses a SOAP-based network protocol, the SOAP XRPC protocol. Network communication in XRPC uses XML messages over HTTP. There is ubiquitous support for URIs, specifically HTTP networking, and XQuery engines are perfectly equipped to process XML messages. Moreover, an XML-based message protocol makes it trivial to support passing values of any type from XDM. The choice for SOAP brings as additional advantages seamless integration of XQuery data sources with web services and Service Oriented Architectures (SOA) as well as AJAX-style GUIs.

1.1.2 Extending XQuery with Query Shipping

For efficient processing of XQuery queries in our target environments, the first task is to extend XQuery with a query shipping model.

By default, the XQuery 1.0 standard already allows querying XML documents distributed over the Internet, using a data shipping model. The built-in function \( \text{fn:doc()} \) fetches an XML document from a remote peer to the local server, where it subsequently can be queried. The recent W3C Candidate Recommendation XQuery Update Facility (XQUF) introduces a built-in function \( \text{fn:put()} \) for remote storage of XML documents, which again implies data shipping. In P2P settings such a data shipping model has several serious drawbacks [91, 92, 180]. It is highly inefficient in terms of network latency. For instance, aggregation queries on huge remote XML documents that produce only small results incur large network costs, because complete documents have to be transferred to queries’ local peers. This directly leads to poor scalability both in sizes and in the number of remote documents involved in a query. Bad load balancing is another drawback, because all query execution happens locally, i.e., at the query originator, missing possibilities of exploiting query processing capabilities of remote peers.

There have been various proposals to equip XQuery with a query shipping model, especially the function shipping style distributed querying abilities [74, 134, 144, 172]. In this research work, we choose to extend XQuery with a Remote Procedure Call (RPC) mechanism, which we call XRPC (i.e., XQuery RPC). On the syntax level, we consider XRPC an incremental development of the existing extensions, with specific advantages concerning simplicity and optimisability. Considering simplicity, XRPC adds RPC to XQuery in the most simple way: adding a destination URI to the XQuery equivalent of a procedure call (i.e., function application). XRPC is optimisable in that it makes explicit the input data (parameters) of a remote query and its result type through the function signature, routinely identified during query parsing in existing XQuery systems. Also, functions can be defined in XQuery modules, and compiled separately in advance, making it easy to do query plan caching and thus accelerate distributed query processing.
1.1.3 Efficiency

In P2P settings (e.g., WAN), query execution times are mainly determined by network latency. The study in [79] demonstrates on a real system that, in distributed systems, the key to scalability is minimising the number of messages. Therefore, in the design of XRPC, we have paid special attentions to minimize the number of messages exchanged among peers and also the sizes of messages. Two new concepts have been introduced: Bulk RPC and runtime XML projection.

**Bulk RPC** The XRPC extension allows maximal flexibility of making XRPC calls. An XRPC call may be placed anywhere in an XQuery query, where a normal XQuery function call is allowed. For example, XRPC calls can be included in for-loops, which often contain large numbers of iterations. Clearly, a naive implementation that handles XRPC calls one-at-a-time will not scale. The SOAP XRPC protocol supports a so-called Bulk RPC concept, which allows the system to compute multiple applications of the same function (with different parameters) in a single request/response network interaction. Bulk RPC is much more efficient than repeated single RPC as network latency is amortized over many calls, and performance becomes bounded by network bandwidth or CPU throughput (hardware factors that scale much better than network latency). Another way to look at Bulk RPC is that it exposes bulk execution opportunities, such that e.g. a function that selects with a constant argument is turned into a join against the sequence of all arguments. Bulk RPC thus has a direct correspondence with set-oriented processing as offered by query algebras, and we believe it can be generally applied in any algebraic XQuery implementation.

**Runtime XML Projection** XML projection[24, 60, 66, 125, 52, 31, 56, 83, 61, 111] is a technique popularly used, e.g., by streaming systems, to reduce the amount of data that needs to be processed for a query and to reduce memory usage. The basic idea of XML projection is, for a given XQuery query $Q$ and an XML document $\mathcal{D}$, to extract a smaller part $\mathcal{D}'$ of $\mathcal{D}$, which is used to execute $Q$ such that $Q(\mathcal{D}) = Q(\mathcal{D}')$. A projection technique usually conducts a compile-time path analysis on $Q$, to derive a set of XPath expressions $P$ that over-estimate the nodes that $Q$ touches. Then, a loading algorithm applies $P$ on $\mathcal{D}$ (from a file or a stream) to generate the projected document (or stream) $\mathcal{D}'$, which is queried with $Q$.

XML projection is also extremely interesting for distributed XQuery processing, as we will see in Chapter 5, where we introduce a new concept called runtime XML projection. When sending XML nodes, pruning huge subtrees, which will remain untouched at the remote sites, can strongly reduce network bandwidth usage, as well as serialisation and deserialisation effort. Our runtime XML projection technique has as additional advantages, compared with compile-time techniques, higher accuracy and flexibility.

Considering accuracy, with the runtime technique, we are able to restrict the projected document using the selection predicates found in expressions, thus, the resulting projected document $\mathcal{D}'$ is often much smaller. For instance, consider the expression //person[@id=$pid]. A compile-time technique, lacking the ability to execute the predicate “@id=$pid”, has to keep all person elements in $\mathcal{D}'$, while our runtime technique only projects the person element, whose id attributed matches the given $\$pid$.

---

2 Note that Bulk RPC should not be confused with semi-join, although they both aim at improving efficiency. Bulk RPC achieves this by reducing the impact of network latency, while semi-join reduces the amount of shipped data.

3 Also called XML filtering or XML pruning.
Runtime projection is more flexible, because it can handle XPath steps on all axes and the built-in functions that need to access XML nodes outside the subtree of their parameters (i.e., fn:root(), fn:id(), fn:idref() and fn:lang()). While handling non-downwards XPath steps (e.g., parent and ancestor) is a cumbersome task for many streaming systems, the runtime projection technique makes it easy. Applying a parent step on a document at runtime is only marginally different from applying a child step.

Thanks to the runtime projection technique, we can stick to a copy-based, stateless approach in the design of XRPC, which minimise network interactions (hence the impact of WAN latency).

1.1.4 Stateless versus Stateful

When designing the basic SOAP XRPC protocol and its extensions, we made a conscious choice for a by-value parameter passing mechanism. That is, during remote function execution, the calling peer (i.e., query originator) will send a request message containing a deep-copy of the parameters to a remote peer, which executes the subexpression, and sends back a response message containing a deep-copy of the result. If the function parameters or results contain XML node-typed items, only the subtrees rooted at these items are serialised (i.e., deep-copied) into the XRPC messages. As a result of this, node-typed items lose their original node identities and structural properties, when they are exchanged among peers, which may affect the semantics of XQuery execution on such shipped nodes. This is an inevitable situation, because when XML nodes must be shipped over the network, it means that, unless one chooses to ship the entire XML document in order to preserve all structural relationships (which defeats the purpose of function shipping), pieces/snippets of the XML document must somehow be copied into the messages.

To illustrate the challenges of distributing XQuery, yet preserving XML node identity, consider a subexpression \( f(a, b) \) with two parameters \( a \) and \( b \) of type \( \text{node()} \), that is executed remotely. Complications may arise, for instance, if the subexpression \( f() \) tests structural XML relationships among its parameters, such as \( a/parent::b \) is \( b \). Similar complications arise when transporting result values back over the network, and when the results of two different function calls to the same remote peer end up at the same peer. It therefore depends on the characteristics of the subexpressions \( f() \), as well as on the way parameters are marshalled in and out of the network messages, whether the distributed query will behave correctly, that is, whether the distributed query is identical to local execution (blindly copying all parameters into the message does not work in this example).

One could consider a simple “callback” way of handling XQuery distribution by not sending XML snippets at all, but just some (global) node identifiers. Each time when a peer needs to execute node-specific XQuery/XPath expressions on such node identifiers, this alternative approach would communicate with the peer where the nodes originally came from, executing the node-specific expressions on that peer and returning the results. While such an approach circumvents semantic problems, it has many drawbacks: (i) it basically gives up on the desire to move computation to more powerful peers; (ii) it introduces additional network round-trips; (iii) it makes all distributed queries – even read-only queries – stateful: a single query might consist of multiple (potentially many) network requests and the query processor on each peer must keep a session context open to guarantee repeatable reads consistency, which causes extra memory consumption and lock contention; and finally (iv) additional protocols would be needed to properly terminate such stateful distributed queries, adding extra protocol com-
plexity, bookkeeping overhead and network latencies. In contrast, the techniques introduced with XRPC lead to flexible query distribution where subexpressions can be moved to the peer that can most efficiently process them. Typically, each peer is visited only once, thus network interactions are minimised and peers can handle the subqueries in a *stateless* manner.

Let us have a more precise look at the difference between a callback approach (i.e., stateful) and the XRPC approach (i.e., stateless). XRPC leads to $O(P_Q)$ number of network round trips, where $P_Q$ is the number of different documents opened by the query $Q$. This is obviously the minimal number of network round trips. A naive callback approach leads to $O(P_Q \times (X_Q + 2B_Q))$ network round trips, where $X_Q$ is the number of XPath steps in the query $Q$, and $B_Q$ the number of binary operators (e.g., $\text{is}$, $\ll$, $\text{union}$, etc). Although it is possible to reduce the number of network round trips caused by a naive callback approach, what we are trying to make clear here is that, in this research, the question is not so much whether XRPC is better than a callback approach, but whether the XRPC approach is possible. This is a major research question.

Additionally, one could also envision a network protocol that combines “callback” query processing with our techniques, something that might be interesting for handling distributed updating queries (because in the default XQUF semantics only locally stored documents can be updated, hence one would have to “callback” to the peer where a node originated from, to apply the update actions). However, given our target of Internet-wide P2P query processing with high network latencies, we decided against the “callback” approach in our own prototype construction, and fully focused on XQuery execution by moving XML snippets and computations on them over the network. In this research work, we have solved the problem brought by this approach in preserving semantic correctness, and also demonstrate the efficiency of this approach (Chapter 5).

### 1.2 Research Objective

The focus of this thesis is processing and optimisation of distributed XQuery queries. The general research question addressed here 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?*

This question identifies three main issues: *efficiency*, *interoperability*, and *scalability*. To better understand the research question, we have refined it into more specific questions:

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

   Determined by the main research question, such an extension should provide the potential for efficient, interoperable and scalable XQuery processing. Additionally, the extension should be orthogonal to all XQuery features, and it should be kept as simple as possible. By allowing *any* kind of XQuery expressions to be executed remotely, the extension provides maximal possibilities for remote execution, which in turn provides more potential for query optimisation. A simple design might look easy, but it is important for efficiency (think of administrative overhead), interoperability (i.e., easy to understand) and scalability (an extension requiring minimal administration is usually much faster than a complex one).
2. **How can different XQuery engines be united to jointly evaluate a single query?**

A single XQuery query could easily involve multiple remote XML documents served by peers that are possibly capable of processing XQuery queries. In general, the best way to handle such queries is to exploit the query processing power on the remote peers, i.e., executing subexpressions of a query on remote peers close to data sources. An open protocol is a prerequisite for heterogeneous XQuery engines to communicate with each other.

3. **How are distributed updating queries supported?**

With the introduction of XQUF, XQuery is no longer a read-only language. Since XRPLC is designed to be an orthogonal extension of XQuery, it also allows updating expressions to be executed on remote peers. This requires a clear definition of the semantics of distributed updating queries (e.g., updates on remote documents), which isolation levels are supported, and how distributed transactions are supported.

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

Decomposing queries to address multiple data sources is a well-studied optimisation problem in relational [175], object-oriented [115, 105], and semi-structured databases [166, 167]. While it is natural to assume that many of the existing techniques can be carried over, the XML data model and the XQuery language introduce a number of particular challenges not met elsewhere, that revolve around XML node identities and structural (rather than value-based) relationships between nodes. For this reason, automatic XQuery decomposition must determine which subexpressions can be decomposed in order to guarantee the correctness of the decomposed queries.

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

Both DBMS and P2P networks are mature research fields, thus, instead of inventing a P2P DBMS from scratch, our strategy for advancing the state-of-the-art in distributed DBMS is to research how to couple existing XDBMS with Distributed Hash Table based P2P overlay networks: which information should the underlying DHT overlays provide to the XDBMS, and how should this information be exposed? How can the DHT overlays benefit from the data management and query processing features supported by the XDBMS, to offer a finer grained data sharing feature than file-based data sharing, and more powerful searching facilities than keyword based search?

### 1.3 Thesis Outline

This thesis is further organised as follows. We start with a discussion of related work in Chapter 2.

In Chapter 3, we introduce the XRPLC language extension, which adds a Remote Procedure Call (RPC) mechanism to XQuery. First, we specify the XRPLC syntax and the SOAP XRPLC network communication protocol. Then, we spend considerable time in rigorously defining the formal semantics of read-only as well as updating XRPLC calls. Finally, we discuss the implementation of XRPLC in MonetDB/XQuery, including the correspondence of Bulk RPC with the loop-lifting technique applied by the pathfinder compiler. This chapter addresses the research questions 1 and 2, and is based on the following paper:
1.3. THESIS OUTLINE


In Chapter 4, we discuss various uses of XRPC for distributed XQuery processing on heterogeneous XQuery engines. First, by using Saxon, we demonstrate how XRPC can be used already with any XQuery system, using an XRPC wrapper that is capable of translating Bulk RPC requests into XQuery. We also show how XRPC can be used to elegantly express various distributed query processing strategies, including experiments in which MonetDB/XQuery and Saxon work together over XRPC, using e.g. the distributed semi-join strategy. Then, we turn our attention to the interaction between XRPC and XQUF. We first define a deterministic distributed update semantics and show that a small extension to the SOAP XRPC protocol enables the protocol to conform to the deterministic update semantics. We then describe how the industry standard Web Service Atomic Transaction[55] could be adapted to support atomic distributed commits of XQUF queries on heterogeneous XQuery engines. Finally, we discuss our first step towards integrating an XDBMS and DHT-based overlays in MonetDB/XQuery* . While XRPC already allows performing P2P queries, it still misses a number of vital P2P functionalities (robust connectivity, peer and resource discovery, approximate query/transaction processing). In Section 4.6, we propose two different ways to couple an XDBMS with DHTs, i.e., a loose-coupling and a tight-coupling. This chapter addresses the research questions 2, 3 and 5, and is based on the following papers (in the order of their appearance):


In Chapter 5, we present an XQuery decomposition framework and three decomposition algorithms that automatically decompose any XQuery query into subqueries for remote execution under different parameter passing semantics. We start with identifying the semantic differences of remote XQuery pass-by-value function evaluation with respect to standard, local function evaluation. We then describe an XQuery Core based query decomposition framework. This leads in to a conservative XQuery decomposition strategy that avoids semantic problems simply by refraining from decomposition in all problem cases. To make our rewrites more effective and robust against syntactic variation, we also describe normalisation and code motion rewrite strategies. To broaden the possibility of query distribution, we
extend the pass-by-value semantics with a new pass-by-fragment message format that conserves more structural relationships between nodes passed in a message, and allows more predicates to be distributed. The pass-by-fragment semantics is subsequently refined to a pass-by-projection semantics by means of a novel runtime XML projection technique, which we use to generate messages that conserve all needed structural relationships between transferred XML nodes, and thus allows even more freedom in query decomposition. As a runtime technique, it is able to prune XML data much more than previously described compile-time projections[52, 31, 111]. Then, we discuss how updating queries can be handled, both in the normal XQUF semantics, as well as under an extension in which we allow non-local documents to be updated. Finally, we give an evaluation of the performance benefits of our techniques in the context of MonetDB/XQuery. This chapter addresses the research question 3 and it is based on the following papers:


- Y. Zhang, N. Tang, P. A. Boncz. Projective Distribution of XQuery with Updates. IEEE Transactions on Knowledge and Data Engineering, 2010. To appear.4

In Chapter 6, we formally prove the correctness of the three decomposition algorithms proposed in Chapter 5. We first prove, for each algorithm, that executing allowed subexpressions of a query remotely over XRPC will produce the same results as the original query, under the XQuery deep-equal semantics. Then, we prove the correctness of the algorithms on XCore queries containing XQUF expressions, and the correctness of the distributed code motion technique.

In Chapter 7, we illustrate how the described P2P XDBMS in Chapter 4 can be used in StreetTiVo. StreetTiVo is a demo application for multimedia meta-data for the so-called Home Theatre PCs. We first describe the current system architecture of StreetTiVo, and its major components, beside XRPC, ASR and PF/Tijah. This first version of StreetTiVo was chosen to be simple, so that we could quickly demonstrate the cooperation between XRPC, ASR, PF/Tijah in a non-trivial application setting, thus a distributed architecture is adopted, in which all StreetTiVo users (i.e., clients) are managed by a central server. Then, we explain the envisioned P2P model and discuss the challenges on our way to make StreetTiVo a truly P2P application. This chapter presents some preliminary ideas to address the research question 4, and is based on the following paper:


Finally, we conclude the thesis and discuss future work in Chapter 8.

---

4The extended version of the ICDE 2009 Conference paper accepted by the TKDE for its Special Issue on the Best Papers of ICDE2009, scheduled for publication in 2010.