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

Link to publication

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.
This Ph.D. work is related to a large body of research works in the area of query processing, query optimisation and transaction management. For more literature, we would like to refer the readers to the surveys [178] and [114]. The books [57] and [135] give comprehensive overviews of techniques involved in all aspects of distributed DBMSs. In this chapter, we first discuss in Section 2.1 related research work in Peer Data Management Systems (PDMSs), with focus on distributed XML querying. Then, we discuss in Section 2.2 two particular techniques that are often used for efficient distributed (XML) query processing, namely XML document filtering and query decomposition.

2.1 P2P Data Management Systems

The tremendous size of the Internet and the popular use of P2P applications have presented new challenges to distributed DBMS researchers. Several visionary papers and surveys have been published that identify new issues in data management in large-scale heterogeneous P2P systems, and introduce possibilities or survey the state of the art on PDMS.

In [86], Gribble et al. raise the question how data management can be applied to P2P and what the database community can learn from it and how they can contribute. The authors discuss the challenges of data placement in P2P environments and introduce the Piazza system [103], a peer data management system that enables sharing heterogeneous relational data using schema mapping. In [33], Bernstein et al. introduce the Local Relational Model (LRM) to also address issues in data management in P2P environments. LRM assumes that the data residing in different databases have semantic inter-dependencies, and allows peers to specify coordination formulas that explain how the data in one peer must relate to data in an acquaintance. Thus, LRM does not require a global schema. Sung et al. give a nice survey of data management in P2P systems in [168]. This paper gives a comprehensive overview of existing unstructured and structured P2P network systems. It also discusses the data integration issues that P2P systems must deal with when sharing structured data, query processing techniques in both P2P file sharing and data sharing systems, and data consistency issues that the cache manager and update manager must address when data duplication is supported. Bonifati et al. [44] carefully compare the characteristics of P2P databases with those of distributed, federated and multi-databases to gain better insight to the P2P data management technology. The authors provide a taxonomy of the most prominent research efforts toward the realisation of full-fledged P2P DBMSs for relational data. [99] is another extensive survey of the state of the art of PDMS. The authors examine open research directions in several areas of PDMS,
including system model, semantics, query planning schemes and maintenance.

Among the material published, [113] is a survey dedicated to distributed processing of XML data in P2P systems. It focuses on data management issues including indexing, clustering, replication and query processing and routing. In [113], P2P DBMSs are classified based on (i) the degree of decentralisation, (ii) the topology of the overlay network, (iii) the way information is distributed among the nodes, and (vi) the type of data they store.

In the remainder of this section, we first describe other XQuery extensions that equip XQuery with a query-shipping model (Section 2.1.1). We then discuss related work on distributed XML query processing in P2P systems. Proposals are grouped by the topology of their underlying overlay networks, i.e., structured P2P systems (Section 2.1.2) and unstructured P2P systems (Section 2.1.3). Finally, in Section 2.1.4, we discuss some prominent work of PDMSs of relational data.

### 2.1.1 Extending XQuery with Query-Shipping

**XQueryD** XQueryD [144, 29] extends XQuery with a new statement “execute at”, which supports remote execution of free-form XQuery queries (specified in the “xquery { ... }” block following an “execute at” statement). XQueryD uses a runtime rewriter to scan the XQuery expressions in the `xquery` block for variables and to substitute them with the runtime values. XQueryD has two main features that distinguish it from other XQuery extensions.

XQueryD is the only XQuery (query-shipping) extension that has a mechanism to explicitly catch and handle exceptions. For each `xquery` statement, one can define multiple `handle` clauses to specify how to react to certain exceptions. The necessity of such an error handling mechanism in XQuery has been confirmed by the recently published W3C Working Draft XQuery 1.1 [149], which proposes to add `try...catch` expressions to the XQuery language.

The primary goal of the XQueryD project is to develop tools to help neuroscientists understand language organisation in the brain. In this context, data sources using totally different data models must be integrated. Therefore, XQueryD also allows an “execute at” statement to be used to invoke queries in a foreign language, other than XQuery.

**Galax XButler and the Yoo-Hoo! Service** Galax Yoo-Hoo! [72] is a user-presence service that uses XButler [134] to integrated user-presence information from multiple service providers and to provide Yoo-Hoo! clients with information such as “the requested subscriber is on-line at Jabber” or “the subscriber’s cell phone is busy”. XButler extends XQuery with a new statement “import service” to import WSDL [62] modules, which enables access to Web services from within XQuery queries:

```xml
import service namespace foo="http://example.org" name "UserProfile";
let $c := foo:getContact("user1", 4) return ($c/name, $c/tel)
```

In the above query, the call to `foo:getContact()` will trigger a SOAP call to the appropriate Web service. Parameter values and results are passed between the caller and the callee using SOAP RPC messages. Thus, once a service module is imported, all operations defined in this service module could be called in an XQuery query, as if they were normal XQuery functions.

When a WSDL module is imported, each operation defined in this module is compiled into an XQuery stub function with the same name and parameters as the WSDL operation. The stub function takes care of generating SOAP RPC request messages with the actual parameter values, exchanging messages with the Web service, and extracting results from the SOAP RPC response message. XButler also provides a tool `xquery2soap` to deploy a SOAP service
from a given XQuery module. In [134], a binding between XQuery and WSDL is given which is based on the relationship between XQuery modules and WSDL portTypes.

Related to XButler is the work proposed by Fourny et al. in [78, 77], which also adds a Web service importing feature to XQuery to allow (asynchronous) accessing of Web services. In [78, 77], the authors have extended JavaScript to allow execution of XQuery expressions (including updating expressions) in Web browsers. Such an extension could ease the development of AJAX-style applications, because programming the browsers involves mostly XML navigation and manipulation, and the XQuery family of W3C standards were designed exactly for this purpose.

**DXQ** DXQ [74, 70] is a distributed XQuery processing framework developed as an extension of Galax. The basic idea of DXQ is that functions in an imported module can be executed on an arbitrary number of remote DXQ servers. DXQ allows remote updates, by means of supporting XQueryP [59]. A small change in module importing is that DXQ distinguishes a module’s interface from its implementation, thus, in DXQ, multiple servers can export the same module interface, but each provides its own, potentially unique implementation. Next to synchronous remote executions, DXQ also provides an asynchronous execution model. This useful feature opens more opportunities for (distributed) query optimisation.

A major difference between DXQ and other XQuery extensions is that DXQ depends on distributed query plans, in terms of the internal Galax execution algebra, generated by the Galax optimiser. This has certain advantages, such as better control over the capabilities of the distributed nodes and possibly better physical plans and optimisation, but the use of an internal algebra makes it hard to achieve cross-system DXQ.

### 2.1.2 Distributed XML Querying in Structured P2P Systems

We start with a detailed discussion of the Active XML project and its related KadoP system in a dedicated subsection, because it is by far the most important related work in distributed processing of XML data that has achieved notable results [28, 19, 20, 165, 12, 18, 5, 9, 36, 154, 11, 16, 37, 15, 14, 4, 6, 8, 13, 30, 155, 10, 3, 127, 138, 7].

#### 2.1.2.1 Active XML and KadoP

**Active XML** Active XML (for short: AXML) is a declarative framework that uses Web services (WSDL) for distributed XML data management. The framework is based on AXML documents, which are XML documents that may contain embedded calls to Web services; and AXML services, which are Web services capable of exchanging AXML documents. An AXML peer is a repository of AXML documents that can act both as client when invoking the embedded service calls, and as server when providing AXML services. A comprehensive overview of AXML features can be found in [9]. In [18, 20], the authors give a formal analysis of the behaviour of AXML systems, in particular, to verify the temporal properties of AXML documents. In this section, we highlight several AXML aspects that are closely related to our work.

A service call in an AXML document is denoted using a special axml:call element, which has one service attribute to specify the service to be invoked. An axml:call element can also contain additional attributes to specify, for instance, the mode, in which the call is activated, and for how long its results are valid. Parameter values of the called services are included as the descendants of the call element [9, 30]. In an AXML document, axml:call
elements are allowed to appear anywhere in the document. This is not a problem for XML documents without schemas, as AXML documents are syntactically valid XML documents. However, adding axml:call elements to an XML document with a schema would either invalidate the original document, or require the schema to be modified.

After a service call has been invoked, its results are materialised, i.e., added into the document. The service call could either be replaced by its results, or be kept next to its results for re-evaluation, which enables data refreshing and nested calls. Service functions are defined in AXML using an XML query language X-OQL[22], in the open-source implementation [28], which itself does not allow distributed evaluation (an embedding piece of AXML is always needed). The SOAP protocol used for AXML uses a document/literal encoding to represent XML subtree values. However, it has not been specified formally.

AXML has shown the value of distributed query optimisation, identifying lazy evaluation schemes and various rewrite strategies [6]. AXML has also shown its value in many different areas of distributed XML processing, including mobile systems [138], data warehousing [13, 14, 5], event systems [4], and workflow systems [164, 165].

**KadoP**
An important application of AXML in P2P systems is KadoP [13, 14], a distributed infrastructure for warehousing XML resources in a P2P framework. KadoP builds on DHTs as a peer communication layer and AXML as a model for constructing and querying the resources in the network. A KadoP query is a tree pattern whose nodes represent data items and whose edges represent containment relationships among the nodes. Both nodes and edges are indexed in the underlying DHT network. The usefulness of KadoP has been demonstrated by the EDOS system [11, 17] and the WebContent platform [5]. EDOS is a distribution system for efficient dissemination of open-source software through the Internet using a publish-subscribe infrastructure. WebContent is a platform for managing distributed repositories of XML and semantic Web data. In [12], the authors describe techniques to address some of the scalability limitations of KadoP. The authors introduce a technique based on partitioning and distributing index blocks to greatly reduce query response time. Structural Bloom Filters are used to reduce network traffic by filtering out peers that surely do not have certain XML nodes needed by a query.

### 2.1.2 XPath Query Processing over DHTs

A considerable number of proposals on supporting XPath queries over DHT overlays already exist, e.g., Galanis et al. [79], XP2P [46, 45], XPeer [142], Skobeltsyn et al. [161] and XCube [120]. These proposals are similar to each other, in the sense that they all focus on efficient processing of the ‘/’ and ‘//’ XPath steps, with minor differences in additionally supported XPath language features. The work proposed by [79, 46, 142, 120] support selection predicates, and the work proposed by [142, 161] support the wildcard ‘∗’. In addition, XCube supports tag-based queries, e.g., (title, “Database”). These proposals differ mainly in the used indexing techniques, which, in turn, affect the way queries a processed. To the best of our knowledge, there is no existing system that supports full-fledged XQuery over DHTs.

**Galanis et al.** In [79], XML tag names are used as hash keys. The authors try to address the scalability issue in large-scale P2P systems by sending queries only to the repositories that have data relevant to the query without relying on a centralised catalog infrastructure. Therefore, the authors propose a catalog framework that is distributed across the data sources themselves. A fully decentralised catalog service allows data providers to join and make their
data query-able by all peers in the systems. To balance query workload, catalog information is split or replicated dynamically.

**XP2P** XP2P [46, 45] is built on top of Chord [162]. The system assumes that each peer stores a set of XML fragments. Each fragment is unambiguously identified by its distinct linear absolute path. The search key of a fragment is the hash value of its path. To compute the hash values, a special fingerprinting technique is used which produces shorter hash keys (than those produced by Chord) and supports a concatenation property that allows the computation of the tokens associated with path expressions to proceed incrementally. XPath queries that only contain child steps (potentially with positional predicates) can be answered in XP2P extremely fast, since the actual query can serve as the search key in the DHT network. For descendant steps, a separate algorithm is presented that achieves a $O(N_f \times \log(N))$ complexity, where $N_f$ is the number of fragments and $N$ the number of peers in the network.

**XPeer** XPeer [142]\(^1\) maintains indices at three different levels of granularity, i.e., DHT hashes of complete XML documents, XPath expressions, and XML elements. Although these indices might accelerate query execution, the authors did not discuss the costs of creating and maintaining the indices.

**Skobeltsyn et al.** Similar to XP2P, Skobeltsyn et al. [161] also index XPath steps containing only the child axis ‘/’. The P-Grid [1, 2] DHT overlay network is used, which uses a binary trie topology. The hash key of a query is the longest sequence of element tags in the query that are only divided by ‘/’. If only one peer is responsible for the hash key, the query result can be computed. Otherwise, a shower algorithm is used to broadcast the query to all peers in the subtrie defined by the hash key. To overcome the high costs imposed by large broadcasts (when the hash key is short), intermediate results are cached.

**XCube** XCube [120] is a tag-based scheme that manages XML data in a hyperCube overlay network to support XPath and tag-based queries. An advantage of tag-based queries is that they do not require users to know the structure of a document before querying the document. Like XPeer [142], documents in XCube are indexed at different levels of granularity. That is, an XML document is compactly represented as a triple: a bit vector derived from the distinct tag names in the document, a synopsis of the document and a bit map of the content summary. A query is processed in four phases. First, the bit vector derived from the query tags is used to locate the query’s anchor peer, which contains a superset of the synopses of all potentially matching answers. Second, the query is compared against all synopses at its anchor peer and forwarded to the anchor peer of each document with matching synopsis according to its bit vector. Third, at the anchor peer of a document, the predicates in the query are examined based on the bit maps stored on this peer. Documents that satisfy the structural requirements but not the predicates in the query are pruned. Finally, the query is forwarded to all owner peers in the answer set for evaluation. Unlike previous approaches, e.g., [79, 46, 45, 142, 161], XCube does not put any limitations on the supported XPath expressions. Regrettfully, the experiment results do not show how well XCube would perform when processing XPath steps on reverse and horizontal axes.

\(^1\)This XPeer project should not be confused with another XPeer project proposed by Sartiani et al. [156], which supports the FLWOR expressions of XQuery on top of a superpeer network.
2.1.3 Distributed XML Querying in Unstructured P2P Systems

**Mutant Query Plans** Similar to AXML, query execution in Mutant Query Plans (MQPs) [136, 137] also happens by exchanging XML documents containing both resulting XML data and unevaluated subqueries, and it is also independent of any central coordinators. However, exchanging (system specific) query plans would suffer from the same interoperability problem as we have pointed out in DXQ. In an XML query language, the usual way to reference data source is to use their URLs. An MQP is a query plan graph serialized in XML that, in addition to URLs, can refer to abstract resources using URNs and include verbatim XML data. In a system using MQP, each peer maintains a local catalog that maps each URN to either a URL, or to a set of peers that know more about this URN. When a peer receives an MQP, it first resolves all URNs in the MQP it knows about. Then, the peer (re)optimises the plan and creates subplans that can be evaluated locally, with associated costs. Next, the peer’s policy manager will decide to accept or reject the mutant plans (e.g., costs are too high), and how much of a (sub)plan can be evaluated locally. Finally, the peer substitutes each evaluated subplan with its results, as an XML fragment, to get a new mutated query plan. If the plan is not yet fully evaluated, the peer chooses the next peer to forward the plan to, by consulting its local catalog. Otherwise, the plan is the final result of the query (in the form of an XML document, without any URNs) and it is forwarded to the query’s destination, which may be different from its origin.

**XPeer** XPeer [156]² is a P2P XDBMS for sharing and querying XML data on top of a superpeer network. Peers export a tree-shaped DataGuide description of their data that is automatically inferred by a tree search algorithm. The query language supported is the FLWR subset of XQuery without universally quantified predicates and sorting operations. Query compilation is performed in two phases by the superpeers. First, the peer that issues the query translates it into a location-free algebraic expression. Then, the query is sent to the superpeer network for the computation of a location assignment. After the location assignment is completed, the query is sent back to the peer that issued it for execution to minimize the load of the superpeer network. The peer applies common algebraic rewriting and then starts query execution: the query is split into single-location subqueries that are sent to the corresponding peers. Subqueries are locally optimized and the results are returned to the initial peer, which executes operations such as joins involving multiple sources. The query algebra of XPeer takes data dissemination, data replication and data freshness explicitly into account.

**HePToX** HePToX [43, 42] is a heterogeneous P2P XDBMS that supports a subset of XQuery. A key idea is that whenever a peer enters the system, it provides a mapping between its schema and a small number of the existing peer schemas. The peers chosen by the entering peer are called its acquaintances. Although (semi-)automatic schema mapping tools could be used to provide the mappings, HePToX also allows a peer database administrator to supply simple correspondences between the peer’s schema and the schemas of its acquaintances. The correspondences are used as a basis for automatically inferring a mapping expression, expressed in the form of Datalog-like rules. HePToX implements a more expressive extension of Global-Local-as-View (GLaV) mappings, called data schema interplay, where mappings exploit correspondences between attribute values and names of schema elements.

---
²This XPeer project should not be confused with another XPeer project proposed by Rao et al. [142], which is an XML-based content query system built on DHT systems.
Piazza Piazza [86, 95, 94, 93, 171, 103] is a well-known PDMS that enables sharing and integration of heterogeneous data. It can handle the mapping of both relational data [95, 171] and XML data [103]. Piazza assumes that participating peers are willing to share their data and define pairwise mappings between their schemas. In [103] Tatarinov et al. describe several methods for optimising schema-based reformulation of XML queries. In [103], data is represented in XML, peers schemas in XML Schema, and mappings are described as query expressions using a subset of XQuery. Peers are considered as connected through semantic paths of such mappings. Peers may store mappings, data or both. Instead of a local index, each peer maintains mappings between its own schema and the schemas of its immediate neighbours. Query evaluation is incremental with an additional logical-level search where data are located based on schema-to-schema mappings. Query processing starts at the issuing peer and is reformulated over its immediate neighbours, which, in turn, reformulate the query over their immediate neighbours and so on. Whenever the reformulation reaches a peer that stores data, the appropriate query is posed on that peer, and additional results may be appended to the query result. Various optimisations are considered regarding the query reformulation process such as pruning semantic paths based on XML query containment, minimizing reformulations and pre-computing some of the semantic paths.

Bremer et al. [51] introduces a distribution approach for a virtual XML repository. XML data are fragmented based on a global conceptual schema and allocated among peers. Fragments allocation is done using existing allocation models for relational databases [26, 135]. The information of fragments allocation, i.e., which fragments are allocated at which peer(s), is kept in a global context. Query processing is done by shipping index entries among nodes and evaluating chains of local joins of indices. By using the global context, the peer, on which a query is started, can compute the remote peers that contain fragments needed by the query.

Koloniari et al. In [112], the authors propose a content-based approach to route XPath queries in a hierarchical P2P network. In this network, peers with similar content are clustered together. Each peer maintains two types of filters: a local filter summarising the documents stored locally at the peer, and one or more merged filters summarising the documents of the peer’s neighbours. These filters are used to route a query only to those peers that may contain relevant documents. Two multi-level Bloom filters are proposed for summarising hierarchical data which exploit the structure of data. Like many proposals for processing XPath queries over DHTs discussed in Section 2.1.2, the work in [112] only supports the child and descendant XPath steps. The proposed approach is more effective for simple linear XPath expressions, but not precise for finding answers for descendant axes. [112] is one of the few approaches that considers updates. When a document is updated, inserted or deleted at a peer, the peer updates its local filter and propagates the updates to merged filters on other peers that use this local filter. The propagation algorithm ensures that only the changed parts of the multi-level filter are transferred, not the whole filter.

Distributed Evaluation of Semistructured Data [167] is one of the first works about evaluation of path expressions on distributed (though non-P2P) semistructured data. In this work, semistructured data is modeled as a rooted, labeled graph, and queries are regular path expressions with complex data restructuring and subqueries. Distribution is implemented by having links from the local XML data to XML objects at remote peers. The model distinguishes between local links that point to local objects and cross-links that point to remote objects. Every peer determines which of its data have incoming edges from other peers (input data
nodes) and which have outgoing edges to remote objects (output data nodes). Given a query, an automaton is computed and sent to every node. Each node traverses only its local graph starting at every input data node and with all states in the automaton. When the traversal reaches an output data node, it constructs a new output data node with the given state. Similarly, new input data nodes are also constructed. Once the result fragments, which consist of an accessibility graph that has the input and output data nodes and edges between them, are computed they are sent to the origin of the query. The originating peer of the query assembles these fragments by adding missing cross-links, and computes all data nodes accessible from the root. The algorithms guarantee that the size of the data exchanged depends only on the number of cross links and the size of the query answer.

2.1.4 PDMSs of Relational Data

Much research work has been done on querying relational data in P2P settings. In this section, we will discuss some prominent work in this area.

**UniStore** UniStore [107, 108] is a triple storage system built on top of the P-Grid [2] DHT overlay, which is based on the ideas of a Universal Relational Model and the Resource Description Framework (RDF). It proposes a structured query language Vertical Query Language (VQL), which is derived from SPARQL [139]. Query plans are processed in a similar way as the Mutant Query Plans [136, 137].

**Hyperion** Hyperion [27, 150, 186] is a PDMS built on top of its own unstructured P2P network, which supports SQL queries over heterogeneous relational data. Like UniStore, Hyperion avoids the need of a global schema by defining mappings between acquaintances. However, unlike UniStore (which requires that acquaintances must be defined at the time a peer enters the system), acquaintances are formed dynamically at runtime, and it does not use a database administrator. Mappings are defined in so-called *mapping tables* which specify relations between values of attributes of data records residing on different peers rather than on schema elements. Distributed Event-Condition-Action (ECA) rules are used to enable and coordinate data sharing.

**PeerDB** PeerDB [132] is built on top of the BestPeer network [131], a two layer hierarchical network that integrates mobile agents. Thus, PeerDB adopts mobile agents to assist in query processing. On each peer, a MySQL database is used to manage data. PeerDB proposes a mechanism to share data of similar but different schemas. It uses a simple data integration algorithm with some user interference and it relies on a central directory server.

**The APPA System** The APPA (Atlas Peer-to-Peer Architecture) system [23, 126, 174] provides high-level data sharing services in large-scale distributed environments by combining Grid and P2P technologies. The architecture of APPA consists of three layers. The *P2P Network* layer provides network independence with services that are common to different P2P networks. Thus, APPA can combine different P2P networks (e.g., JXTA, Chord and CAN) to exploit their relative advantages. The *Basic Services* layer provides elementary services, e.g., persistent data management, communication cost management and group membership management. The *Advanced Services* layer provides advanced services for semantically rich data sharing including schema management, replication, query processing, security, etc., using the basic services.

**System P** In System P [152, 151], query plans are computed decentralised, locally at the peers. Based on the given Local-as-View (LaV) and Global-as-View (GaV) peer mappings, a
local rule-goal tree is created at the peer receiving the original query, as well as at every peer that is contacted during query processing. System P balances the completeness of the query result and execution cost by pruning the query plan at mappings that are estimated to yield only few result tuples. For query execution, the authors propose a budget driven approach, where peers are assigned a budget to use for query answering. This is similar to the economic execution model of Mariposa [163].

2.2 Related Query Processing Techniques

In this section, we discuss related work in two specific techniques for efficient distributed (XML) query processing, namely XML document filtering and query decomposition.

2.2.1 XML Document Filtering

XML document projection or filtering is a technique popularly used by both streaming systems and main memory XQuery processors to drastically reduce the size of the data model representation, which in turn can accelerate query processing.

Marian et al. [125] originally introduce this concept and propose a static analysis algorithm to compute, at compile time, for a given XQuery query the set of projection paths which include a set of used paths and a set of returned paths. Before the query is evaluated, a load algorithm uses the projection paths to compute projected documents, which could be much smaller than the original documents. The query is then applied on the projected documents instead. This compile-time XML projection technique is used to address memory limitations in main memory XQuery processors. Our runtime XML projection technique [183, 184] (Chapter 5) is an extension of the compile-time XML projection. Instead of only supporting XPath steps on downward axes (e.g., self, child and descendant), the runtime technique supports XPath steps on all axes and the built-in functions fn:root(), fn:id(), fn:idref() and fn:lang() which need to access nodes outside the subtree of their node parameters. Comparing with the compile-time projection technique, runtime projection is much more accurate, because predicates on XPath steps are processed before the projection. In such cases, the resulting projected documents of runtime projection are usually much smaller than the results of compile-time projection.

Much research work has been done on filtering XML documents in streaming systems for efficient query processing [24, 60, 66, 52, 31, 56, 83, 61, 111]. We will take a look at several of these approaches. XFilter [24] aims at efficient matching of XML documents to large numbers of user profiles that are expressed as XPath queries. It is the first approach that uses a Finite State Machine for each XPath query to quickly locate relevant profiles when an XML document arrives. Based on XFilter, Diao et al. propose YFilter [66] that combines all XPath queries into a single Nondeterministic Finite Automaton. Bressan et al. [52] introduce a precise XML pruning technique for a subset of XQuery FLWOR expressions, based on the a priori knowledge of a data guide for underlying XML data. However, it does not handle XPath predicates, backward axes and XQuery-like languages. A type-based XML projection technique [31] is studied to improve current solutions with comparable or higher precision and less pruning overhead, and supporting backward XPath axes. This technique is only applicable for XML documents that have a DTD. Koch et al. [111] also propose runtime XML projection techniques. Based on the static compilation of runtime lookup-tables and a runtime automaton from projection paths and a DTD, an input XML document can be filtered efficiently using
string matching algorithms. This technique improves efficiency, but still lacks the power of supporting reverse XPath axes and XQuery built-in functions. This reflects a common limitation of the XML filtering techniques: they only support efficient processing of subsets of XPath.

### 2.2.2 Query Decomposition

Decomposing queries to address multiple data sources has been applied in a large variety of research areas, including relational databases [175, 106], object-oriented databases [34, 105, 115], distributed databases [98, 119], multi-databases [117], heterogeneous distributed databases [173, 123], P2P data management systems [33], and semi-structured databases [166, 167]. Decomposition techniques have been proposed based on ontologies [177], topics (i.e., Topical Query Decomposition) [40, 169], and (hyper-)trees [76, 157, 82].

Proposals exist that study decomposing XML queries, but none of them address the problems that revolve around XML node identity and structural (rather than value-based) relationships between nodes when queries are decomposed and distributed automatically. In [166, 167], the author discusses the decomposition of unstructured query languages on a semi-structured database (a rooted, labeled graph). In [118], Le et al. propose a bottom-up approach for distributed XDBMSs using Global-As-View to transform a global XPath expression into local XPath expressions executable in local schemas. This approach requires structural information about peers to supervise decomposition. In [160], Silveira et al. present a query decomposition mechanism that allows a query stated at the conceptual level, using CXPath (a Conceptual XPath language defined by the authors), to be decomposed into an XQuery statement at the XML level. Other works in distributed XML query evaluation, such as [53, 63, 170], only focus on a restricted set of XQuery/XPath queries, and do not address the problem of transparent query decomposition, such that these challenges do not play a role.

For instance, in [170], the authors consider optimizing the cost of communication in answering XPath queries over distributed data based on the client-server model. Minimal views that contain results of a single query or a set of queries are used to avoid the redundancy met in such results where the same data may appear many times. The system leaves part of the evaluation of the query to the client that may have to extract all the answers from the minimal view to obtain the results to the initial queries. At a receiving peer (i.e., the client side), the system ensures that only downwards XPath steps will be applied on the received data, avoiding problems around XML node identities and structural relationships between nodes that can be caused by executing non-downward XPath steps (e.g., \textit{ancestor} and \textit{preceding}) on the received data.

### 2.3 Conclusion

From the related research work, we can conclude that the topic of distributed XML querying has attracted quite some research interest. Within this area, two major approaches have been studied: \textit{i}) extending the XQuery standard with a query shipping model, and \textit{ii}) accelerating of XPath queries on distributed XML documents. XQuery/XPath has been generally accepted as the language to query XML data. DHT is a popularly used mechanism to manage underlying networks, because DHT networks have several properties (e.g., $O(\log N)$ scalability) that make them particularly suitable for P2P settings. However, plenty of issues are still left open.
This Ph.D. research addresses three open issues. First of all, interoperability is an almost untouched topic by any of the existing proposals, while it is a main issue in P2P settings, where peers are highly heterogeneous. Although XButler uses the standard SOAP RPC protocol as its communication protocol, due to the limitations of SOAP RPC, it is restricted to only support XQuery functions with atomic value parameters and results. This in turn reduces the interoperability of XButler. Secondly, supporting full-fledged XQuery is an open issue. Most of the existing work only focuses on efficient distributed processing of small subsets of XQuery/XPath. This greatly reduces the expressive power of the XQuery language and also the interoperability of the proposed techniques. Moreover, with XQUF [58], XQuery is no longer a read-only language. This raises several questions, such as what are the semantics of remote updates, how can distributed transactions be supported, which consistency level should be provided and how can it be integrated into the language. Thirdly, the issue of dealing with challenges imposed by the XML data model is unaddressed. Query decomposition is an often used mechanism in distributed query processing. However, the existing techniques were developed for relational data, which are only value based, while in the XML world, XML nodes have node identities and structural properties. It has not been studied if and how the existing techniques can be applied to the XML data model, while respecting the semantics of XML data.