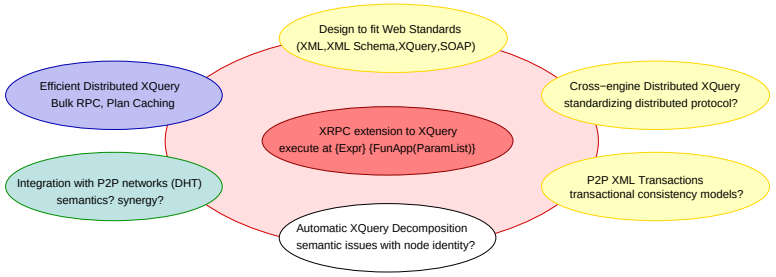
XRPC – Interoperable and Efficient Distributed XQuery

Ying Zhang Peter Boncz
CWI Amsterdam, The Netherlands



XRPC

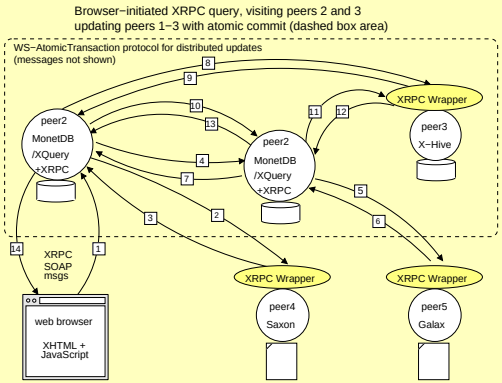
is a **minimal XQuery extension** that enables **distributed** yet **efficient** querying of **heterogeneous** XQuery data sources. XRPC enhances the existing concept of XQuery functions with the Remote Procedure Call (RPC) paradigm. By calling out of an XQuery for-loop to multiple destinations, and by calling functions that themselves perform XRPC calls, complex P2P communication patterns can be achieved. The XRPC extension is **orthogonal** to all **XQuery features**, including the **XQuery Update Facility**.

XRPC is also a **network SOAP sub-protocol**, that integrates seamlessly with web services and Service Oriented Architectures (SOA), and AJAX-based GUIs. A crucial feature of the protocol is **Bulk RPC** that allows remote execution of many different calls to the same procedure, using possibly a single network round-trip. The **efficiency potential** of XRPC is demonstrated via an open-source implementation in MonetDB/XQuery. We show, however, that XRPC is **not system-specific**: every XQuery data source can service XRPC calls using a wrapper.

Since XQuery is a pure functional language, we can leverage techniques developed for functional query decomposition to rewrite data shipping queries into XRPC-based function shipping queries. Powerful **distributed database techniques** (such as **semi-join optimizations**) directly map on bulk RPC, opening up interesting future work opportunities.

XRPC + XQuery Update Facility = Distibuted/P2P transactions

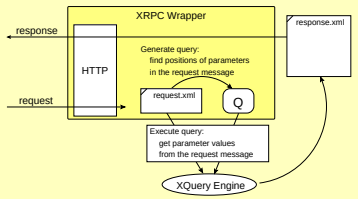
- ACID & P2P? Currently up to Repeatable Reads. More possible?
- 2PC + SOAP: existing standard WS-AtomicTransaction (IBM et al.)



Cross-engine XRPC via Wrapper

- Saxon, Galax & X-Hive tested
- only for handling calls

Architecture of the XRPC Wrapper



Query generated for the filmsByActor() XRPC request

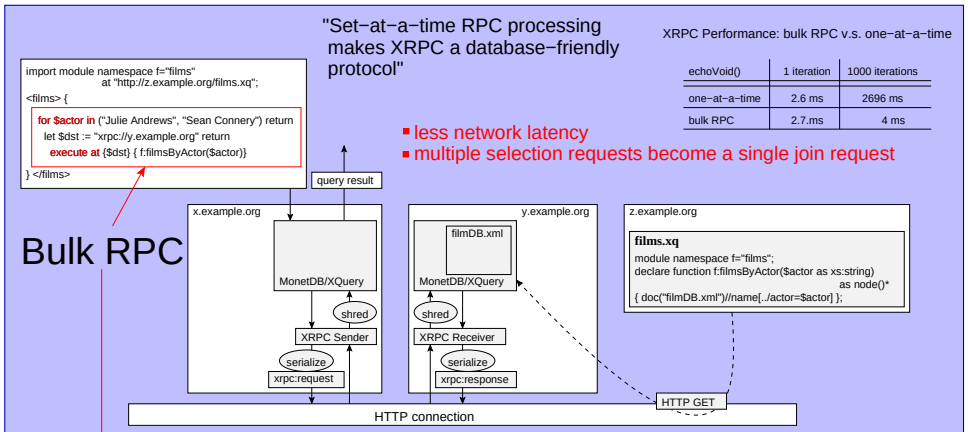
```
import module namespace f="films" at "http://z.example.org/films.xq";
<env:Envelope ...>
 <env:Body>
  <xrpc:response xrpc:module="films" xrpc:method="filmsByActor"> {
    for $call in doc("/tmp/requestXXX.xml")//xrpc:call
    let $param1 := n2s($call/xrpc:sequence[1])
    return s2n(f:filmsByActor($param))
  } </xrpc:response>
 </env:Body>
</env:Envelope>
```

"Set-at-a-time RPC processing makes XRPC a database-friendly protocol"

XRPC Performance: bulk RPC v.s. one-at-a-time

echoVoid()	1 iteration	1000 iterations
one-at-a-time	2.6 ms	2696 ms
bulk RPC	2.7.ms	4 ms

- less network latency
- multiple selection requests become a single join request



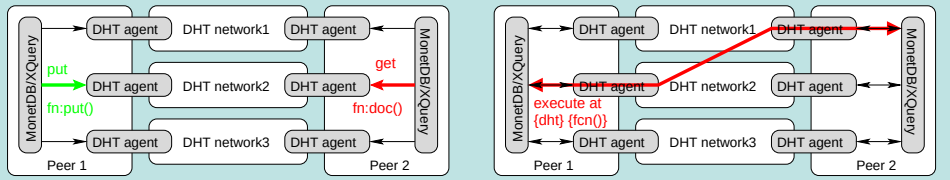
SOAP XRPC Request message of the call "execute at {$dst} { f:filmsByActor($actor) }

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:xrpc="http://monetdb.cwi.nl/XQuery"
              xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:xs="http://www.w3.org/2001/XMLSchema"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://monetdb.cwi.nl/XQuery http://monetdb.cwi.nl/XQuery/XRPC.xsd">
 <env:Body>
  <xrpc:request xrpc:module="films" xrpc:method="filmsByActor" xrpc:arity="1" xrpc:location="http://z.example.org/films.xq">
   <xrpc:call><xrpc:sequence><xrpc:atomic-value xsi:type="xs:string">Julie Andrews</xrpc:automic-value></xrpc:sequence></xrpc:call>
   <xrpc:call><xrpc:sequence><xrpc:atomic-value xsi:type="xs:string">Sean Connery</xrpc:atomic-value></xrpc:sequence></xrpc:call>
  </xrpc:request>
 </env:Body>
</env:Envelope>
```

multiple function calls at once (set-wise execution)

XRPC & Distributed Hash Tables (DHT)

- DHTgoodies: robustness against churn, log(N) hop lookup, automatic data replication
- How to integrate; syntax? semantics? synergy DHT stats and distributed (X)Query optimization?



Automatic Decomposition and Query Distribution

- **Goal**: automatically decompose a data-shipping XQuery query into subqueries to execute each subquery at (or close to) the peer owning the (XML) data.
- **Challenge**: **distributed node identity**, i.e., how can we preserve nodes' identities when they are copied (as parameter or as results of the subqueries) among peers, so that the distributed execution won't alter the query's original semantics.
- **Idea**: **Pass-By-Projection**
  - Ship ancestors/siblings of a node to remote peers, if those will be needed
  - Do not copy a node's descendants if those node will not be used.
  - Runtime XML Projection:
    - higher precision that compile-time XML projection
    - all XPath axes, including reverse axes such as ancestor
    - built-in functions, e.g., fn:id() and fn:idref()