Highly Distributed XQuery with DXQ

Mary Fernandez Trevor Jim
AT&T Labs Research
{mff,trevor}@research.att.com

Nicola Onose
UCSD
nicola@cs.ucsd.edu

Kristi Morton
University of Texas Austin
kmorton@cs.utexas.edu

Jerome Simeon
IBM T.J. Watson Research
simeon@us.ibm.com

ABSTRACT

Many modern applications, from Grid computing to RSS handling, need to support data processing in a distributed environment. Currently, most such applications are implemented using a general purpose programming language, which can be expensive to maintain, hard to configure and modify, and require hand optimization of the distributed data processing operations. We present Distributed XQuery (DXQ), a simple, yet powerful, extension of XQuery to support distributed applications. This extension includes the ability to deploy networks of XQuery computing nodes, to remotely invoke XQuery programs on those nodes, and to ship code between nodes. We demonstrate the capabilities of DXQ by building a network of name servers that capture the functionality of DNS, the Domain Name System. We show that our system can flexibly accommodate different patterns of distributed computation, and discuss some simple but essential distributed optimizations.

1. INTRODUCTION

Distributed data management represents the next frontier for database technology. Many modern applications, from Grid computing to RSS handling, require infrastructure that goes beyond the traditional client-server paradigm upon which traditional database management systems are built. They promise many benefits, like high performance by harnessing the power of under-utilized networked resources, reliability by leveraging the inherent redundancy of resources in a network, and concurrency by distributing computation simultaneously over multiple resources. Such promises cannot be fulfilled, however, without reliable infrastructure for configuring and managing heterogeneous components in a distributed system. The goal of the DXQ project is to support the development of *reliable*, *extensible*, and *efficient* distributed, data-intensive, applications.

DXQ We present Distributed XQuery (DXQ), a simple, yet powerful, extension of XQuery to support distributed applications. This extension includes the ability to deploy networks of XQuery computing nodes, the ability to call XQuery programs on those nodes, and to ship code between nodes. In this demonstration, we illustrate the capabilities of DXQ by building a version of the Domain Name System (DNS) in which name servers are implemented by DXQ processes and use XML to represent DNS resource records. DNS is representative of more complex distributed applications, e.g., for resource management [5, 11], or multicast [4]. Those protocols are usually implemented in a general purpose programming language, which make them expensive to maintain and hard to optimize. In addition to DNS, we hope to demonstrate an implementation of the Narada protocol [4] which we are currently building using DXQ.

With more and more of the data processed in such distributed environment being XML, we believe DXQ provides a natural and effective framework for these applications. The DXQ code for DNS resolution is less than 100 lines of code, while a typical C implementation is 5000+ lines of code. DXQ thus enables rapid development and makes programs easier to modify and maintain. DXQ also implements distributed query optimization with minimal developer effort. Finally, DXQ enables dynamic optimization not only of queries but also of the location at which queries will execute.

Demonstration Our demonstration introduces DXQ's distributed features, shows how they can be used to support highly distributed applications in a simple and maintainable way, and illustrates some simple but essential distributed optimizations. We use DNS as a motivating example. For demonstration purposes, the various nodes on the network will be represented as separate processes running on the same machine, and communicating over sockets. We will demonstrate several scenarios of increasing complexity:

Iterative DNS Resolution. Resolution is the process of looking up an address for a hostname by consulting various nameservers; in DXQ, it is easily accomplished with a simple iterative query.

Optimized Iterative Resolution. We improve performance through distributed optimization, notably, combining multiple queries and pushing joins from client to server.

Recursive Resolution. We show how to simply implement a different distribution pattern in which the servers recursively call each other, off-loading work from the client.

Caching Resolution. This scenario demonstrates how XML updates [3] can implement caching for DNS servers.

Multicast. This is the most advanced scenario, where we demonstrate how DXQ can distribute a query to execute in parallel on many nodes in the network. This requires asynchrony and is essential to scale such applications to a large number of nodes.

DXQ is implemented on top of the Galax XQuery 1.0 processor, and it is publicly available as part of Galax's lat-

SIGMOD'07, June 11–14, 2007, Beijing, China.
ACM 978-1-59593-686-8/07/0006.

est release. Due to space limitations, this proposal only briefly describes the code and features of the DNS application. A complete description of the various scenarios that will be demonstrated at the conference, the corresponding DXQ code, and graphical traces that illustrate the application running, can be found at: `http://db.ucsd.edu/dxq/`.

Related Work Most of the work on distributed databases [10] focuses on distributed query processing in a controlled environment with a limited number of nodes. Work in the area of peer to peer networks (e.g., [13]) focuses on algorithms for efficient distribution of queries and is complementary to our work, which focuses on the query distribution infrastructure. DXQ's remote execution operator was inspired by our previous work on using distributed databases for security policies [6, 7]. XQueryD [12] extended XQuery with exception handling and an operator identical to our `exec`; DXQ currently lacks exception handling, but include asynchronous `exec` and XML updates. Sophia [14] is an extension of Prolog with a feature like our `exec`. In comparison to DXQ, Sophia includes continuous update, but lacks asynchronous execution and XML support. NDlog [8] and SeNDlog [1] support distributed execution through recursive queries. Unlike DXQ, NDlog and SeNDlog require specified link relations between nodes. This restriction mirrors the physical needs of network but limits the expression of relational data.

2. DXQ OVERVIEW

The DXQ language includes all of XQuery 1.0, along with updates [2, 3] and extends it in two ways. To the query prolog, it adds an import-interface declaration, which permits a client query to remotely call functions in a module exported by a DXQ server. To the expression language, it adds a remote-execute expression in the style of [12], which redirects evaluation of an expression to a DXQ server. The following grammar gives the syntax for these extensions, which are illustrated in the next section.

Prolog	::=	$\cdots$ | `import interface namespace` *Prefix* `= URI;`
Expr	::=	$\cdots$
	|	`bind` $Prefix_1$ `as` $Prefix_2$ `at` $Expr_1$ `return` $Expr_2$
	|	`exec at` *Prefix* `return` *Expr*
	|	`exec at` *Prefix* `do` *Expr*

The `bind` expression is used to dynamically bind an interface to its implementation on a given machine (specified by the expression after the `at` clause). The implementation is identified by a new namespace prefix ($Prefix_1$) which is in scope for the expression after the `return` clause ($Expr_2$). The two `exec at` expressions are used to ship a query (*Expr*) to a given location synchronously (with the `return` clause), or asynchronously (with the `do` clause). A synchronous exec evaluates to the result of the remote execution, while an asynchronous exec returns the empty sequence immediately.

2.1 DXQ Architecture

Much like a Web server exports an interface of permissible messages, a *DXQ server* exports a module of XQuery functions that may be called remotely from a client application or from another DXQ server. Unlike a Web server, a DXQ server can evaluate *any* XQuery expression submitted by a requester, which may include calls to the server's exported functions. The ability to move arbitrary queries between DXQ servers enables distributed query optimization, which can reduce the number of messages between servers and reduce the amount of data moved between servers that are necessary to evaluate a query. In response to a request, a DXQ server may evaluate an expression and return an *extensional* XML value. In addition, a DXQ server may return an *intensional* XQuery expression that the requester may evaluate to compute the result itself. This mechanism enables a flexible range of resource-management *policies* to be applied in a distributed system.

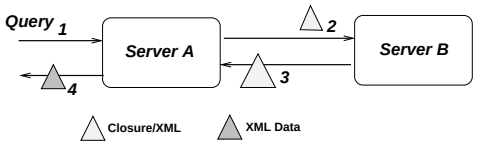

Figure 1: DXQ Servers

Figure 2.1 shows the basic interaction of two DXQ servers, which can be generalized to any number. Server A accepts an XQuery expression from a user application (Step 1), compiles that query given the server's module context, evaluates the resulting query, and returns an XML value (Step 4). To compute the result may require that some fragment of the compiled query be evaluated by Server B, in which case Server A constructs a closure that encapsulates the query fragment and ships the closure to Server B (Step 2). Similarly, Server B compiles the closure in its own context and returns a result to Server A (Step 3).

3. IMPLEMENTING DNS WITH DXQ

Due to space limitations, we only illustrate parts of the first and fourth scenarios listed at the beginning of this proposal. DNS [9] provides the basic infrastructure for name resolution on the internet, mapping a hostname to its IP address. DNS name servers are organized as a distributed tree. Each name server exports a database of *resource records* that contains the IP address of a hostname or a delegation indicating that another nameserver is responsible for the IP address of a hostname. In DXQ, every peer server first exports the following XQuery module interface, which contains one function that returns all of a server's resource records:

```
module namespace Server = "DNSServer";
declare function Server:RR() external;
```

The `Server:RR` function returns local DNS resource records. For example, the module below is the start of authority for research.att.com, and includes the hostname and IP address for the "root" nameserver:

```
module namespace Server = "DNSServer";
declare function Server:RR() {
<rr>
 <soa dom="research.att.com" srv="ns.research.att.com"/>
 <a host="www.research.att.com" addr="192.20.3.54"/>
 <ns dom="." serv="a.root-servers.net"/>
 <a host="a.root-servers.net" addr="198.41.0.4"/>
</rr>
};
```

With a network of servers in place, organizations can run a DNS *resolver* which can contact known servers in order

to resolve names requested by users. Using DXQ, the DNS resolver algorithm can be implemented by the following recursive function:

```
 1. import interface Server = "DNSServer";
 2. import module Util = "DNSUtility";
 3. declare function Resolver:lookup($x,$n) {
 4.  <rr>{
 5.   let $rr := exec { $x } { Server:RR() }
 6.   return
 7.    $rr/a[@host=$n],
 8.    (for $ns in $rr/ns, $a in $rr/a
 9.       where $ns/@srv=$a/@host
10.       and fn:not($ns/@dom = ".")
11.       and Util:name-lt($ns/@dom,$n)
12.     return Resolver:lookup($a/@addr, $n)/a)
13.  }</rr>
14. };
```

The query contains one `exec` expression (Line 5), and the server X returns all its resource records to the requester. The result includes all address records for host H defined by server X (Line 7), and the address records defined by nameservers known to X (Lines 8–12), which requires calling `lookup` recursively (Line 12). This particular implementation is iterative, i.e., one resolving server contacts all the delegating servers until one of them returns an IP address for the name, but the servers do not contact each other. A more complex scenario is for the servers to contact each other through recursive distributed calls and propagate the result back to the resolving server.

3.1 DNS with caching

Updates in XQuery [3] can be used to support advanced features, such as caching. For example, to avoid unnecessary computation, the following version of `lookup` maintains a cache of hostname, address pairs, indexed by server:

```
1. declare variable $cache := <cache/>;
2. declare updating function Resolver:lookup($x,$n) {
3.  <rr>{
4.   let $ac := $cache/entry[a/@host=$n] return
5.   if (empty ($ac)) then (
6.    let $rr := exec{ $x }{ Server:RR() }
7.    let $ans := $rr/a[@host=$n]
8.    return
9.      (if (empty($ans)) then ()
10.       else
11.        do insert
12.         <entry srv="{$x}">{$ans}</entry>
13.         into $cache },
14.       $ans,
15.    ... Lines 8-12 above ...)
16.   else $ac
17.  }</rr>
};
```

Note that Lines 4–13 first check whether the name requested is in the cache and if it is not, executes a lookup and updates the cache. XQuery with updates can also support more advanced caching policies. At the conference, we plan to demonstrate other DNS features as well, such as expiration of cache entries, batch lookup of hostnames, and conditional protocols that depend on the requester's IP address. Likewise, we'll present in detail the multicast scenario. All these features are implemented concisely and transparently in DXQ.

4. REFERENCES

[1] M. Abadi and B. T. Loo. Towards a declarative language and system for secure networking. In *NetDB '07: Proceedings of the 3rd International Workshop on Networking meets Databases*, 2007.

[2] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML query language. Technical report, World Wide Web Consortium, Nov. 2006. http://www.w3.org/TR/xquery.

[3] D. Chamberlin, D. Florescu, and J. Robie. XQuery update facility. Technical report, World Wide Web Consortium, July 2006. http://www.w3.org/TR/xqupdate/.

[4] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), 2002.

[5] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The open grid services architecture, version 1.5. Technical Report GFD.80, Global Grid Forum, 2006. http://www.ggf.org/documents/GFD.80.pdf.

[6] C. A. Gunter and T. Jim. Policy-directed certificate retrieval. *Software Practice and Experience*, 30(15):1609–1640, 2000.

[7] T. Jim and D. Suciu. Dynamically distributed query evaluation. In *PODS*, 2001.

[8] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: language, execution and optimization. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 97–108, New York, NY, USA, 2006. ACM Press.

[9] P. Mockapetris. Domain names—concepts and facilities. http://www.ietf.org/rfc/rfc1034.txt, 1987.

[10] M. T. Ozsu and P. Valduriez. *Principles of distributed database systems*. Prentice-Hall, 1991.

[11] L. Peterson and J. Wroclawski. Overview of the GENI architecture. Technical Report Design Document 06-11, Global Environment for Network Innovations, 2006. http://www.geni.net/GDD/GDD-06-11.pdf.

[12] C. Re, J. Brinkley, K. Hinshaw, and D. Suciu. Distributed XQuery. In *Workshop on Information Integration on the Web*, pages 116–121, 2004.

[13] I. Tatarinov et al. The Piazza peer data management project. *SIGMOD Records*, 32(3):47–52, 2003.

[14] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: an information plane for networked systems. *SIGCOMM Comput. Commun. Rev.*, 34(1):15–20, 2004.