DeXIN– An Extensible Framework For Distributed XQuery Over Heterogeneous Data Sources*

Muhammad Intizar Ali[1], Reinhard Pichler[1], Hong Linh Truong[2], and Schahram Dustdar[2]

[1] Database and Artificial Intelligence Group, Vienna University of Technology
{intizar,pichler}@dbai.tuwien.ac.at,
[2] Distributed Systems Group, Vienna University of Technology
{truong,dustdar}@infosys.tuwien.ac.at

Abstract. In the Web environment, rich, diverse sources of heterogeneous and distributed data are ubiquitous. In fact, even the information characterizing a single entity - like, for example, the information related to a Web service - is normally scattered over various data sources using various languages such as XML, RDF, and OWL. Hence, there is a strong need for Web applications to handle queries over heterogeneous, autonomous, and distributed data sources. However, existing techniques do not provide sufficient support for this task. In this paper we present DeXIN, an extensible framework for providing integrated access over heterogeneous, autonomous, and distributed web data sources, which can be utilized for data integration in modern Web applications and Service Oriented Architecture. DeXIN extends the XQuery language by supporting SPARQL queries inside XQuery, thus facilitating the query of data modeled in XML, RDF, and OWL. DeXIN facilitates data integration in a distributed Web and Service Oriented environment by avoiding the transfer of large amounts of data to a central server for centralized data integration and exonerates the transformation of huge amount of data into a common format for integrated access.

Key words: Data Integration, Distributed Query Processing, Web Data Sources, Heterogeneous Data Sources.

1 INTRODUCTION

In recent years, there has been an enormous boost in Semantic Web technologies and Web services. Web applications thus have to deal with huge amounts of data which are normally scattered over various data sources using various languages. Hence, these applications are facing two major challenges, namely (i) how to integrate *heterogeneous* data and (ii) how to deal with *rapidly growing* and continuously changing *distributed data sources*.

* This work was supported by the Vienna Science and Technology Fund (WWTF), project ICT08-032.

The most important languages for specifying data on the Web are, on the one hand, the Extensible Markup Language (XML) [1] and, on the other hand, the Resource Description Framework (RDF) [2] and Ontology Web Language (OWL) [3]. XML is a very popular format to store and integrate a rapidly increasing amount of semi-structured data on the Web while the Semantic Web builds on data represented in RDF and OWL, which is optimized for data interlinking and merging. There exists a wide gap between these data structures, since RDF data (with or without the use of OWL) has domain structure (the concepts and the relationships between concepts) while XML data has document structure (the hierarchy of elements). Also the query languages for these data formats are different. For XML data, XQuery [4] has become the query language of choice, while SPARQL [5] is usually used to query RDF/OWL data.

It would clearly be useful to enable the reuse of RDF data in an XML world and vice versa. Many Web applications have to find a way of querying and processing data represented in XML and RDF/OWL simultaneously. There are several approaches for dealing with *heterogeneous* data consisting of XML, RDF and OWL: The most common approach is to transform all data sources into a single format [6, 7] and apply a single query language to this data. Another approach to deal with heterogeneity is query re-writing which poses queries of different query languages to the data which is left in the original format, thus avoiding transformation of the whole data sources [8]. A major drawback of the transformation-based approaches is that the transformation of data from one language into the other is a tedious and error prone process. Indeed, an RDF graph can be represented by more than one XML tree structure, so it is not clear how to formulate XQuery queries against it. On the other hand, XML lacks semantic information; so converting XML to RDF results in incomplete information with a number of blank nodes in the RDF graph. Moreover, many native XML and RDF data storage systems are now available to tackle rapidly increasing data sizes. We expect in the near future that many online RDF/XML sources will not be accessible as RDF/XML files, but rather via data stores that provide a standard querying interface, while the approach of query re-writing limits language functionalities because it is not possible to compile all SPARQL queries entirely into XQuery. In [8], a new query language is designed which allows the formulation of queries on data in different formats. The system automatically generates subqueries in SPARQL and XQuery which are posed to the corresponding data sources in their native format – without the need of data transformation. A major drawback of this approach is that the user has to learn a new query language even though powerful, standardized languages like XQuery and SPARQL exist. Moreover, this approach is not easily extended if data in further formats (like relational data) has to be accessed.

For dealing with *distributed Web data sources*, two major approaches for query processing exist: centralized query processing transfers the distributed data to the central location and processes the query there, while decentralized query processing executes the queries at remote sites whenever this is possible. With the former approach, the data transfer easily becomes the bottleneck of

the query execution. Keeping replica on the central location is usually not feasible either, since we are dealing with autonomous and continually updating data sources. Hence, in general, decentralized query processing is clearly superior. Recently DXQ [9] and XRPC [10] have been proposed for decentralized execution of XQuery and, likewise, DARQ [11] for SPARQL. However, to the best of our knowledge, a framework for decentralized query execution to facilitate data integration of heterogeneous Web data sources is still missing.

In this paper we present DeXIN (**D**istributed **e**xtended **X**Query for heterogeneous Data **IN**tegration) – an extensible framework for distributed query processing over heterogeneous, distributed and autonomous data sources. DeXIN considers one data format as the basis (the so-called "aggregation model") and extends the corresponding query language to executing queries over heterogeneous data sources in their respective query languages. Currently, we have only implemented XML as aggregation model and XQuery as the corresponding language, into which the full SPARQL language is integrated. However, our framework is very flexible and could be easily extended to further data formats (e.g., relational data to be queried with SQL) or changed to another aggregation model (e.g., RDF/OWL rather than XML). DeXIN decomposes a user query into subqueries (in our case, XQuery or SPARQL) which are shipped to their respective data sources. These queries are executed at remote locations. The query results are then transformed back into the aggregation model format (for converting the results of a SPARQL query to XML, we adhere to the W3C Proposed Recommendation [12]) and combined to the overall result of the user query. It is important to note that – in contrast to the transformation-based approaches mentioned above [6, 7] *only the results are transformed* to a common format. The main contributions of this paper are as follows.

- We present DeXIN – an extensible framework for parallel query execution over distributed, heterogeneous and autonomous large data sources.
- We come up with an extension of XQuery which covers the full SPARQL language and supports the decentralized execution of both XQuery and SPARQL in a single query.
- Our approach supports the data integration of XML, RDF and OWL data without the need of transforming large data sources into a common format.
- We have implemented DeXIN and carried out experiments, which document the good performance and reduced network traffic achieved with our approach.

2 APPLICATION SCENARIO

DeXIN can be profitably applied in any Web environment where large amounts of heterogeneous, distributed data have to be queried and processed. A typical scenario can be the area of Web service management. The number of Web services available for different applications is increasing day by day. In order to assist the service consumer in finding the desired service with the desired properties, several Web service management systems have been developed. The

Service Evolution Management Framework (SEMF) [2] [13] is one of these efforts to manage Web services and their related data sources. SEMF describes an information model for integrating the available information for a Web service, keeping track of evolutionary changes of Web services and providing means of complex analysis of Web services. SEMF facilitates the selection of the best Web service from a pool of available Web services for a given task. Each Web service is associated with different attributes which effect the quality of service. Figure 1 gives an impression of the diversity of data related to a Web service. This data is normally scattered over various data sources using various languages such XML, RDF, and OWL. However, currently available systems do not treat these heterogeneous, distributed data sources in a satisfactory manner. What is urgently needed is a system which supports different query languages for different data formats, which operates on the data sources as they are without any transformations, and which uses decentralized query processing whenever this is possible. Moreover, this system should be flexible and allow an easy extension to further data formats. In fact, this is precisely the functionality provided by DeXIN.

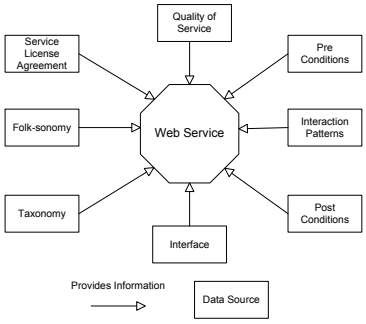

Fig. 1. Data Sources of a Web Service [13]

3 RELATED WORK

Several works are concerned with the transformation of data sources from one language into the other. The W3C GRDDL [6] working group addresses the issue of extracting RDF data from XML files. In [7], SPARQL queries are embedded into XQuery/XSLT and automatically transformed into pure XQuery/XSLT queries to be posed against pure XML data. In great contrast to these two approaches, DeXIN does not apply any transformation to the data sources. Instead, subqueries in SPARQL (or any other language, to which DeXIN is extended in the future) are executed directly on the data sources as they are and only the result is converted. Moreover, in [7], only a subset of SPARQL is supported, while DeXIN allows full SPARQL inside XQuery.

In [8], a new query language XSPARQL was introduced (by merging XQuery and SPARQL) to query both XML and RDF/OWL data. In contrast to [8], our approach is based on standardized query languages (currently XQuery and SPARQL) rather than a newly invented language. Moreover, the aspect of data distribution is not treated in [8].

DXQ[9], XRPC[10] and DARQ[11] are some efforts to execute distributed XQuery and distributed SPARQL separately on XML and RDF data. However,

[2] We acknowledge the assistance of Martin Treiber (Distributed Systems Group ,Vienna University of Technology) for providing access to SEMF data.

the integration of heterogeneous data sources and the formulation of queries with subqueries from different query languages (like SPARQL inside XQuery) are not addressed in those works.

4 DEXIN

4.1 Architectural Overview

An architectural overview of DeXIN is depicted in Figure 2. The main task of DeXIN is to provide an integrated access to different distributed, heterogeneous, autonomous data sources. Normally, the user would have to query each of these

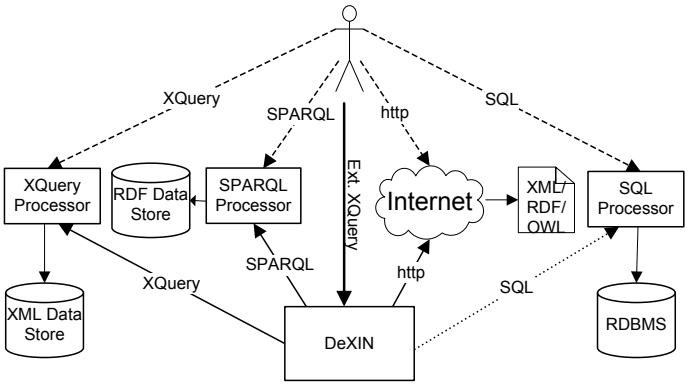

Fig. 2. Architectural overview of DeXIN framework

data sources separately. With the support of DeXIN, he/she has a single entry point to access all these data sources. By using our extension of XQuery, the user may still formulate subqueries to the various data sources in the appropriate query language. Currently, DeXIN supports XQuery to query XML data and SPARQL to query RDF/OWL data. However, the DeXIN framework is very flexible and we are planning to further extend this approach so as to cover also SQL queries on relational data. Note that not all data sources on the Web provide an XQuery or SPARQL endpoint. Often, the user knows the URI of some (XML or RDF/OWL) data. In this case, DeXIN retrieves the requested document via this URI and executes the desired (XQuery or SPARQL) subquery locally on the site where DeXIN resides. DeXIN decomposes the user query, makes connections to data sources and sends subqueries to the specified data sources. If the execution fails, the user gets a meaningful error message. Otherwise, after successful execution of all subqueries, DeXIN transforms and integrates all intermediate results into a common data format (in our case, XML) and returns the overall result to the user. In total, the user thus issues *a single query* (in our extended XQuery language) and receives *a single result*. All the tedious work of decomposition, connection establishment, document retrieval, query execution, etc. is done behind the scene by DeXIN.

4.2 Query Evaluation Process

The query evaluation process in DeXIN is shown in Figure 3. The main components of the framework are briefly discussed below.

Parser. The Parser checks the syntax of the user query. If the user query is syntactically correct, the parser will generate the query tree and pass it on to the query decomposer. Otherwise it will return an error to the user.

Query Decomposer. The Query Decomposer decomposes the user query into atomic subqueries, which apply to a single data source each. The concrete data source is identified by means of the information available in the Metadata Manager (see below). Each of these atomic subqueries can then be executed on its respective data source by the Executor (see below).

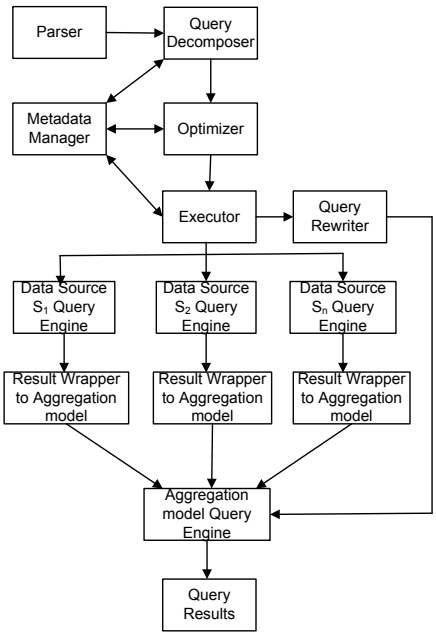

Fig. 3. Query Evaluation Process

Metadata Manager. All data sources supported by the system are registered by the Metadata Manager. For each data source, the Metadata Manager contains all the relevant information required by the Query Decomposer, the Optimizer or the Executor. Metadata Manager also stores information like updated statistics and availability of data sources to support the Optimizer.

Optimizer. Optimizer searches for the best query execution plan based on static information available at the Metadata Manager. It also performs some dynamic optimization to find variable dependencies in the dependant or bind joins. Dependant or bind joins are basically nested loop joins where intermediate results from the outer relation are passed as filter to the inner loop. Thus, for each value of a variable in the outer loop, a new subquery is generated for execution at the remote site. In such scenarios, the optimizer will first look for all possible values of the variables in the outer loop and ground the variables in the subquery with all possible values, thus formulating a bundled query to ship at once to the remote site.

Executor. The Executor schedules the execution sequence of all the queries (in parallel or sequential). In particular, the Executor has to take care of any dependencies between subqueries. If a registered data source provides an XQuery or SPARQL endpoint, then the Executor establishes the connection with this data source, issues the desired subqueries and receives the result. If a registered data source only allows the retrieval of XML or RDF/OWL documents via the URI, then the Executor retrieves the desired documents and executes the subqueries locally on its own site. Of course, the execution of a subqueries may fail,

e.g., with source unreachable, access denied, syntax error, query timeout, etc. It is the responsibility of the Executor to handle all these exceptions. In particular, the Executor has to decide if a continuation makes sense or the execution is aborted with an error message to the user.

Result Reconstruction. All the results received from distributed, heterogeneous and federated data sources are wrapped to the format of the aggregation model (in our case, XML). After wrapping the results, this component integrates the results and stores them in temporary files for further querying by the aggregation model query processor (in our case, an XQuery engine).

Query Rewriter. The Query Rewriter rewrites the user query in the extended query language (in our case, extended XQuery) into a single query on the aggregation model (in our case, this is a proper XQuery query which is executed over XML sources only). For this purpose, all subqueries referring to different data sources are replaced by a reference to the locally stored result of these subqueries. The overall result of the user query is then simply obtained by locally executing this rewritten query.

5 XQUERY EXTENSION TO SPARQL

DeXIN is an extensible framework based on a multi-lingual and multi-database architecture to deal with various data formats and various query languages. It uses a distinguished data format as "aggregation model" together with an appropriate query language for data in this format. So far, we are using XML as aggregation model and XQuery as the corresponding query language. This aggregation model can then be extended to other data formats (like RDF/OWL) with other query languages (like SPARQL). In order to execute SPARQL queries inside XQuery, it suffices to introduce a new function called SPARQLQuery(). This function can be used anywhere in XQuery where a reference to an XML document may occur. This approach is very similar to the extension of SQL via the XMLQuery function in order to execute XQuery inside SQL (see [14]). The new function SPARQLQuery() is defined as follows:

```
XMLDOC SPARQLQuery(String sparqlQuery,URI sourceURI)
```

The value returned by a call to this function is of type *XMLDOC*. The function SPARQLQuery() has two parameters: The first parameter is of type *String* and contains the SPARQL query that has to be executed. The second parameter is of type *URI* and either contains the URI or just the name of the data source on which the SPARQL query has to be executed. The name of the data source refers to an entry in the database of known data sources maintained by the Metadata Manager. If the indicated data source is reachable and the SPARQL query is successfully executed, then the result is wrapped into XML according to the W3C Proposed Recommendation [12].

To illustrate this concept, we revisit the motivating example of SEMF[13] discussed in Section 3. Suppose that a user wants to get information about available Web services which have a license fee of less than one Euro per usage. Moreover,

suppose that the user also needs information on the service license agreement and the quality of service before using this service in his/her application. Even this simple example may encounter the problem of heterogeneous data sources if, for example, the service license agreement information is available in XML format while the information about the quality of service is available in RDF format. A query in extended XQuery for retrieving the desired information is shown in Figure 4. We conclude this section by having a closer look at the central

```
for
  $a in doc("http://SEMF/License.xml")/agreement,
  $b in SPARQLQuery(" SELECT ?title ?ExecutionTime
   WHERE {
?x <http://www.w3.org/2001/sub#title> ?title .
?x  <http://www.w3.org/2001/sub#QoS>   ?ExecutionTime "
          },http://SEMF/QoS.rdf)/result
WHERE
$a/servicetitle = $b/title
AND $a/peruse/amount <= 1
RETURN
<Results>
 <Service>
  <ServiceTitle>{$a/title}</ServiceTitle>
  <Requirement>{$a/requirement}</Requirement>
  <ExecutionTime>{$b/ExecutionTime}</ExecutionTime>
 </Service>
</Results>
```

Fig. 4. An example extended XQuery for DeXIN

steps for executing an extended XQuery query, namely the query decomposition and query execution.

The query tree returned by the Parser has to be traversed in order to search for all calls of the SPARQLQuery() function. Suppose that we have n such calls. For each call of this function, the Query Decomposer retrieves the SPARQL query q_i and the data source d_i on which the query q_i shall be executed. The result of this process is a list $\{(q_1, d_1), \ldots, (q_n, d_n)\}$ of pairs consisting of a query and a source. The Executor then poses each query q_i against the data source d_i. The order of the execution of these queries and possible parallelization have to take the dependencies between these queries into account. If the execution of each query q_i was successful, its result is transferred to the site where DeXIN is located and converted into XML-format. The resulting XML-document r_i is then stored temporarily. Moreover, in the query tree received from the Parser, the call of the SPARQLQuery() function with query q_i and data source d_i is replaced by a reference to the XML-document r_i. The resulting query tree is a query tree of pure XQuery without any extensions. It can thus be executed locally by the XQuery engine used by DeXIN.

6 IMPLEMENTATION AND EXPERIMENTS

DeXIN supports queries over distributed, heterogeneous and autonomous data sources. It can be easily plugged into applications which require such a facility. As a case study, we take the example of service management systems and show how

DeXIN enhances service management software by providing this query facility over heterogeneous and distributed data sources. We set up a testbed which includes 3 computers (Intel(R) Core(TM)2 CPU, 1.86GHz, 2GB RAM) running SUSE Linux with kernel version 2.6. The machines are connected over a standard 100Mbit/S network connection. An open source native XML database eXist (release 1.2.4) is installed on each system to store XML data. Our prototype is implemented in Java. We utilize the eXist [15] XQuery processor to execute XQuery queries. The Jena Framework [16] (release 2.5.6) is used for storing the RDF data, and the ARQ query engine packaged within Jena is used to execute SPARQL queries.

6.1 Experimental Application: Web Service Management

One of the main motivations for developing this framework is to utilize it for service management systems like SEMF [13]. Being able to query distributed and heterogeneous data sources associated to Web services is a major issue in these systems. SEMF stores and manages updated information about all the services listed in this framework. Recall the example use case given in Section 5: We consider a user who requests information about available Web services which have a license fee of less than one Euro per usage. Moreover, the user needs information on the service license agreement and the quality of service. We assume that the service license agreement information is available in XML format while the information about the quality of service is available in RDF format. As we have seen in Section 5, our framework provides the user a convenient way of querying these distributed, heterogeneous data sources at the SEMF platform without worrying about the transformation, distribution and heterogeneity of the data sources involved by issuing the extended XQuery query of Figure 4 to SEMF. The result returned to the user is in XML format and may look like the XML file in Figure 5.

```
<Results>
 <Service>
  <ServiceTitle>WISIRISFuzzySearch</ServiceTitle>
  <Requirement>
   <peruse>
    <payment>
     <amount currency='EUR'> 0.90 </amount>
     <taxpercent code='VAT'>20</taxpercent>
    </payment>
   </peruse>
  </Requirement>
  <ExecutionTime Unit='sec'>17</ExecutionTime>
 </Service>
 <Service>
    .........
 </Service>
   ......
</Result>
```

Fig. 5. Result after Executing the Query shown in Figure 4

6.2 Performance Analysis

In order to analyze the performance of DeXIN, we have conducted tests with realistically large data. Since SEMF is only available as a prototype, the test data available in this context is too small for meaningful performance tests. We therefore chose to use DBPedia (see http://dbpedia.org/) and DBLP (see http://dblp.uni-trier.de/xml/), which are commonly used for benchmarking.

Data Distribution over the Testbed. For the SPARQL query execution over RDF data, we use a subset of DBPedia, which contains RDF information extracted from Wikipedia. This data consists of about 31.5 million triples and is divided into three parts (Articles, Categories, Persons). The size of these parts is displayed in Table 1. The data is distributed over the testbed in such a way that the Articles, Categories, and Persons are stored on different machines. Moreover, we have further split these data sets into 10 data sources of varying size in order to formulate queries with subqueries for a bigger number of data sources. For the XQuery execution over XML data we used DBLP. DBLP is an online bibliography available in XML format, which lists more than 1 million articles. It contains more than 10 million elements and 2 million attributes. The average depth of the elements is 2.5. The XML data is also divided into three parts (Articles, Proceedings, Books), whose. size is shown in Table 2. We distributed the XML data over the testbed such that the Articles, Proceedings, and Books are stored on different machines. As with the RDF data, we also subdivided each of the three parts of the XML data into several data sources of varying size.

Table 1. RDF Data Sources.

Name	Description	# Tuples
RS1	Articles	7.6Million
RS2	Categories	6.4Million
RS3	Persons	0.6Million

Table 2. XML Data Sources.

Name	Description	Size
XS1	Articles	250MB
XS2	Proceedings	200MB
XS3	Books	50MB

Experiments. In the first scenario we consider a set of queries of different complexity varying from simple select-project queries to complex join queries. The queries use a different number of distributed sources and have different result sizes. The results shown are the average values over ten runs. The query execution time is subdivided as

Total Time = Connection Time + Execution Time + Transfer Time

Figure 6 presents the query execution time for a naive centralized approach compared with DeXIN. It turns out that the data transfer time is the main contributor to the query execution time in the distributed environment – which is not surprising according to the theory on distributed databases [17]. DeXIN reduces the amount of data transferred over the network by pushing the query execution to the local site, thus transferring only the query results. We observe that with increasing size of data sets, the gap in the query execution time between DeXIN and the naive centralized approach is widened.

In the second scenario we fix the size of data sources and execute queries with varying selectivity factor (i.e., the ratio of result size to data size) and com-

pare the query execution time of DeXIN with the naive centralized approach. As was already observed in the previous scenario, the execution time is largely determined by the network transfer. Figure 7 further strengthens this conclusion and, moreover, shows that DeXIN gives a better execution time for queries with high selectivity. The results displayed in Figure 7 indicate that DeXIN is much stronger affected by varying the selectivity of queries than the centralized approach. DeXIN is superior to the centralized approach as long as the selectivity factor is less than 90% . Above, the two approaches are roughly equal.

In the third scenario, we observe the effect of the number of data sources on the query execution time. We executed several queries with varying number of sources used in each query. Figure 8 again compares the execution time of DeXIN with the execution time of a naive centralized approach. It turns out that as soon as the number of sources exceeds 2, DeXIN is clearly superior.

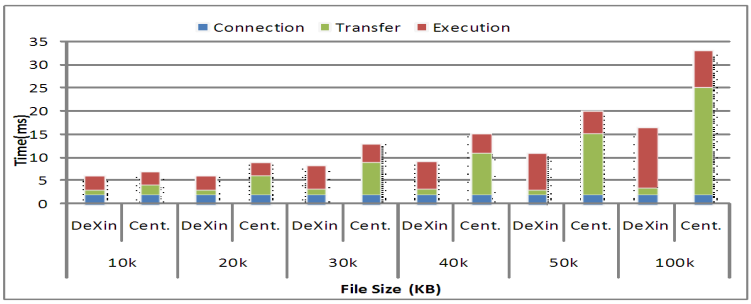

Fig. 6. Execution Time Comparison

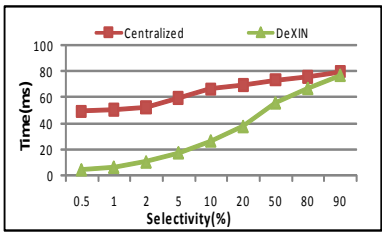

Fig. 7. Varying Selectivity Factor

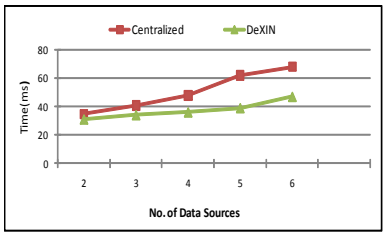

Fig. 8. Varying level of Distribution

7 CONCLUSION AND FUTURE WORK

In this paper, we have presented DeXIN – a novel framework for an integrated access to heterogeneous, distributed data sources. So far, our approach supports the data integration of XML and RDF/OWL data without the need of transforming large data sources into a common format. We have defined and implemented an extension of XQuery to provide full SPARQL support for subqueries. It is worth mentioning that the XQuery extension not only enhances XQuery capabilities to execute SPARQL queries, but SPARQL is also enhanced with XQuery capabilities e.g. result formatting in the return clause of XQuery etc.

DeXIN can be easily integrated in distributed web applications which require querying facility in distributed or peer to peer networks. It can become a powerful

tool for knowledgeable users or web applications to facilitate querying over XML data and reasoning over Semantic Web data simultaneously.

An important feature of our framework is its flexibility and extensibility. A major goal for future work on DeXIN is to extend the data integration to further data formats (in particular, relational data) and further query languages (in particular, SQL). Moreover, we are planning to incorporate query optimization techniques (like semi-joins – a standard technique in distributed database systems [17]) into DeXIN. We also want to extend the tests with DeXIN. So far, we have tested DeXIN with large data sets but on a small number of servers. In the future, when the Web service management system SEMF [13] is eventually applied to realistically big scenarios, DeXIN will naturally be tested in an environment with a large-scale network.

References

1. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition), September 2006. W3C Proposed Recommendation.
2. Dave Beckett and Brian McBride. RDF/XML Syntax Specification (Revised), February 2004. W3C Proposed Recommendation.
3. Deborah L. McGuinness and JFrank van Harmelen. OWL Web Ontology Language, February 2004. W3C Proposed Recommendation.
4. Scott Boag, Don Chamberlin, Mary F. Fernndez, Daniela Florescu, Jonathan Robie, and Jrme Simon. XQuery 1.0: An XML Query Language, January 2007. W3C Proposed Recommendation.
5. Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF, January 2008. W3C Proposed Recommendation.
6. Fabien Gandon. GRDDL Use Cases: Scenarios of extracting RDF data from XML documents, April 2007. W3C Proposed Recommendation.
7. Sven Groppe, Jinghua Groppe, Volker Linnemann, Dirk Kukulenz, Nils Hoeller, and Christoph Reinke. Embedding sparql into xquery/xslt. In *Proc. SAC 2008*, pages 2271–2278, 2008.
8. Waseem Akhtar, Jacek Kopecký, Thomas Krennwallner, and Axel Polleres. Xsparql: Traveling between the xml and rdf worlds - and avoiding the xslt pilgrimage. In *Proc. ESWC 2008*, pages 432–447, 2008.
9. Mary F. Fernández, Trevor Jim, Kristi Morton, Nicola Onose, and Jérôme Siméon. Highly distributed xquery with dxq. In *SIGMOD Conference*, pages 1159–1161, 2007.
10. Ying Zhang and Peter A. Boncz. Xrpc: Interoperable and efficient distributed xquery. In *VLDB*, pages 99–110, 2007.
11. Bastian Quilitz and Ulf Leser. Querying distributed rdf data sources with sparql. In *Proc. ESWC 2008*, pages 524–538, 2008.
12. Dave Beckett and Jeen Broekstra. SPARQL Query Results XML Format, January 2008. W3C Proposed Recommendation.
13. Martin Treiber, Hong-Linh Truong, and Schahram Dustdar. SEMF - Service Evolution Management Framework. In *Proc. EUROMICRO 2008*. IEEE Computer Society, 2008.
14. Jim Melton. SQL, XQuery, and SPARQL: What's Wrong With This Picture? . In *Proc. XTech*, 2006.
15. Wolfgang M. Meier. eXist: Open Source Native XML Database, June 2008.
16. Jena. – A Semantic Web Framework for Java, June 2008.
17. M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.