Sharing Data on the Grid using Ontologies and distributed SPARQL Queries

Andreas Langegger, Martin Blöchl, Wolfram Wöß
Institute of Applied Knowledge Processing
Johannes Kepler University Linz
Altenberger Straße 69, 4040 Linz, Austria
{ martin.bloechl | al | wolfram.woess }@jku.at

Abstract

The vision of the Semantic Web is to make Web content machine-readable. To describe data, the Resource Description Framework has been extended with a schema-level and description logics (e.g. RDF-S and OWL DL). To successfully retrieve data described by RDF-based ontologies, the query language SPARQL is currently being developed at W3C. However, although SPARQL also provides a client/server protocol, currently queries are targeted to single sites only. For a large-scale data integration, which is the scope of a middleware developed within the Austrian Grid project, it is necessary to execute queries on distributed data nodes. Because of the large amount of research undertaken in the field of relational database systems, it is reasonable to apply relational algebra to SPARQL query processors. In fact, most RDF triple stores are based on relational database systems. In this paper, several concepts are proposed to enable the integration of heterogeneous, distributed data sources with SPARQL.

1. Introduction

During the last couple of years, *Grid Computing* has become a new paradigm partly based on existing technologies which have been introduced by the *High-Performance* and *Cluster Computing* communities since the 1960ies. If the *Semantic Web* is about *bringing the Web to its full potential*, Grid Computing would be about bringing all kind of resources connected by the internet to its collaborative potential. Such resources may be CPU power, storage capacity, higher-level services, or available data which is also the key resource of the *Semantic Web* [24]. Several national and transnational Grid projects have been started during the last years, as also in Austria [2]. Especially for scientific collaboration, sharing data between different parties is fundamental. Therefore, data managed by various research communities should also be regarded as Grid resources, shared by Virtual Organizations [12]. To enable a large-scale sharing of scientific data within Grids, a middleware is being developed as part of the Austrian Grid project which will enable the virtual integration of distributed, heterogeneous data sources based on Semantic Web technology.

Scientific data is usually stored in various different storage systems like relational database systems, XML documents, CSV files, or spreadsheets. Sometimes access is available via public Web gateways (simple REST or web service/SOAP endpoints). Additionally, these data sources often use different schemes to store and manage similar data. An integrated global data model for such a scenario will probably become rather complex and moreover, it will be subject of frequent changes and adjustments when new data sources are added or altered. Several key concepts of the Semantic Web are therefore used to enable the integration of a large number of heterogeneous, distributed data sources on the Grid. The key components of the Semantic Web are RDF-based ontologies, whose concepts are referred by URIs and organized by several namespaces. This allows the design of several domain ontologies which are extensible and can be merged and combined in a very flexible way. While the schema level of ontologies can be used to describe the intensional layer for a specific domain (schema), data can be expressed as instances adhering to this schema. Currently, extensible research is done in the fields of ontology modeling, mapping, and evolution as well as model management [22] in general. However, there is still a lack of concepts and testbeds targeted towards systematic retrieval of data on the Semantic Web supporting distributed query execution. To enable the distributed access through a SPARQL query processor, some constraints have to be stipulated which will be discussed in this contribution.

In order to integrate data sources on the Grid, the proposed middleware must be able to map local data models and schemes to several global domain ontologies. As a first approach a formal mapping was defined in two phases [5]. In the first phase local ontologies were created from lo-

1529-4188/07 $25.00 © 2007 IEEE
DOI 10.1109/DEXA.2007.129

cal schemes and afterwards local concepts where mapped to global concepts in the second phase. However, this approach became too complex, because of the heterogeneity in data models, especially when adding support for distributed query processing. As a second step, the mediator-wrapper architecture [14], which is a common method to integrate data sources with heterogeneous data models is used. Because each data source supports different access methods, a dynamic programming algorithm based on the enumeration of rules [19] is applied. Several existing cost models may be integrated for query plan optimization.

2. Related Work

The use of ontologies for data integration is not a new concept. However, there is currently no approach which enables the integration of distributed, heterogeneous data sources by distributing SPARQL query plans. To be usable with up-to-date Grid toolkits, such a middleware must expose OGSA-compliant Grid services, support Virtual Organizations [12], and incorporate existing Grid Security Infrastructure (GSI). Related work can be divided into several domains: RDF triple stores and SPARQL implementations, mapping approaches like D2R-Server [8] or Virtuoso [3], traditional data integration as well as distributed query processing for relational database systems.

Because the storage engine is a core component during query processing, it is important to comprehend how these systems organize data and implement SPARQL. Triple stores manage a large amount of $(subject, predicate, object)$ triples – or (s, p, o) in short – as RDF data. Beyond the abstract triple model of RDF nearly all triple store implementations are using a relational model, mainly because considerable research effort has been done and relational database systems are widely used today. Examples for RDF stores based on relational database systems are the open source projects Sesame [18], Jena [7], 3store [16] as well as commercial products like AllegroGraph [13] and OpenLink Virtuoso [3]. Currently Mulgara [23] (formerly known as Kowari) seems to be the only native RDF store. Some of them [17, 1, 3] support SPARQL, which is likely to become a W3C Recommendation in future. The relational algebra applied to process SPARQL queries has been published for 3store [15] and Jena ARQ [9].

D2R-Server [8] could be used to integrate relational database systems, especially since it is able to translate SPARQL queries to native SQL queries and return tuples as OWL instances. However, a more generic approach, like the mediator/wrapper architecture, is needed, particularly because various storage systems and distributed query processing have to be supported. In order to facilitate these requirements an integrated system has to be developed and therefore the extension or re-implementation of existing SPARQL query processors is required. A possible starting point is the DARQ project [4], started 2006, which is a modification of Jena ARQ to support federated SPARQL queries. Another approach to data integration is GridMiner [6] which is based on a mediator/wrapper approach [21]. However, Gridminer does not use ontologies as global schemes and instead of a formal query language, an abstract query plan is constructed manually by the user. Additionally, it is currently not capable of processing SPARQL queries because of the lack of semantics. However, there is a cooperation with this project within the Austrian Grid. OGSA-DQP [20] is a service-based distributed query processor for Grids which is able to create parallel sub-plans and thus exploit Grid resources more efficiently.

The query processor for the required middleware will be an integrated approach of a distributed and semantic query processor. In this contribution ontology mapping is not discussed. It is assumed that any local data sources already adheres to the global ontology used. In fact various wrappers will translate queries according to the underlaying data model.

3. Architecture Overview

The mediator-wrapper architecture of the system is shown in Figure 1. Clients connect to the mediator and request results by submitting SPARQL queries corresponding to the global domain ontologies. The mediator parses such a query and enumerates multiple query plans using an iterative dynamic programming algorithm [11, p. 48]. The cheapest plan with minimal estimated retrieval time will finally be executed. To overcome model and schema heterogeneity each wrapper provides various specific access methods which will be described in the next section. Usually a wrapper is placed as close as possible to the underlying data sources to benefit of local capabilities like relational join operations. However, some data sources like those available via web service endpoints cannot be extended by a wrapper because there is no direct access to the data source. For those endpoints wrappers may be placed inside a special wrapper container as part of the mediator. To execute global join operations the mediator can temporarily store data inside a local cache. However, based on response time estimates, the query optimizer will chose alternative plans which enable join operations in relational database systems before data is consumed by the mediator.

The catalog which is attached to the mediator contains all global domain ontologies used by the data integration system. The global ontologies are developed in a collaborative process by experts from all scientific domains which will utilize the middleware for their purposes. It may be reasonable to introduce global forums and support collaboration

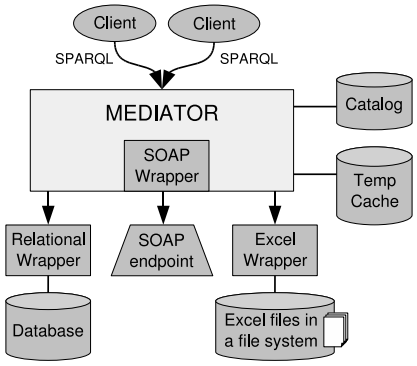

Figure 1. Mediator-wrapper architecture.

with additional tools. When a new data source is registered, the corresponding wrapper may use declarative mappings to some certain extent to facilitate wrapper implementation. However, because of the mentioned complexity, a declarative mapping from an arbitrary data model and schema to the global domain ontologies is often not achievable. This is also the reason why the mediator-wrapper approach is being used now. A wrapper may use mapping information which is specific to the underlaying data model. For instance, an existing relational-ontology mapper like D2RQ [8] can be used which itself applies declarative mappings to rewrite SPARQL to SQL queries.

4. Query Processing

Because relational database systems are the most commonly used information systems, many data integration systems are based on relational algebra. Nearly all RDF storage engines use relational database systems and relational algebra to process SPARQL queries [9]. As shown in [9] a variant of relational algebra can be found for RDF data. The operators for selection, projection, inner join, left outer join (which is required to support `OPTIONAL` expressions), union, and difference as described in [9] have been adopted. Because global ontologies and queries are based on RDF, the system is primarily based on graph pattern matching.

Metadata Catalog The catalog which is attached to the mediator does not only contain the global domain ontologies, additionally information about registered data sources is stored. In order to enable systematic data source selection as part of query optimization, it is necessary for the mediator to know which data source provides which sort of information, or in other words, instances of which classes. Moreover, wrappers may report statistics about the magnitude of instances for each class to improve cost estimation.

Query optimization As mentioned in section 3, query optimization is based on iterative dynamic programming. Firstly, the algorithm generates multiple access plans according to available indices and access capabilities of attached data sources. In a second step, multiple join plans are generated. Especially in a distributed environment there are numerous possible sub-plans for joining data as discussed in [10]. Although the algorithm is commonly used in database systems, it can also be applied to query processing in the mediator-wrapper architecture. The access and join operations are replaced by specific operations provided by wrappers. Again, multiple access and join plans are generated and the cheapest is finally selected. Some sample wrapper operators for a relational data source wrapper and the mediator are:

$$plan\_access(G, F) = R\_Fetch(G, F, D)$$
$$plan\_join(S_1, S_2, V) = R\_Join(S_1, S_2, V) \text{ if } S_1.site = S_2.site$$
$$plan\_join(S_1, S_2, V) = M\_Join(S_1, S_2, V) \text{ if } S_1.site \neq S_2.site$$

G is a graph pattern, F is a set of `FILTER` expressions, D is a data source location, V is a join variable, and S_i are results of sub plans. Results of sub-plans are RDF relations as described in [9]. An RDF relation is a set of RDF tuples and can be represented as a table with query variables as its header and RDF tuples as row data. Data source selection is done by enumerating $plan\_access(G, F, P)$. According to the types in the query and site meta data stored in the catalog, multiple fetch operations with different site locations D are created. Joins inside relational database systems (R_Join) are only possible if the data source of both sub results S_1 and S_2 is the same. Otherwise, the mediator has to execute the join operation M_Join inside the attached cache which will require costly data shipping from S_1 and S_2 to the mediator. If a wrapper does not natively support a join operation, only M_Joins will be possible. The union operator for SPARQL is different from the relational one and is processed as an outer union as defined in [25, Sect. 7]. A `FILTER` expression is always bound to one or more variables and according to [25] it is a restriction on solutions over the whole group in which the filter appears. Hence, the `FILTER` expression is only submitted to relevant graph groups.

Example Based on sample data shown in Tables 1 and 2, a detailed example will be discussed now. The global SPARQL query shown in Listing 1 is used to retrieve sunspot observations (in terms of the global concept `http://gsdam.sf.net/global/science/astro/solar#SunspotObservation`) from July 26–27 2006. More specifically, the query result will contain information about the number of spots and groups observed as well as the lastName and eventually the email address

kso:SO9989	a	sun:SunspotObservation
kso:SO9989	t:inCalendarClockDataType	”2006-07-26T06:00:00Z”
kso:SO9989	sun:groups	1
kso:SO9989	sun:spots	13
kso:SO9989	sun:site	kso:Site
kso:SO9989	sun:observedBy	kso:Scientist18
...		
kso:SO9991	a	sun:SunspotObservation
kso:SO9991	t:inCalendarClockDataType	”2006-07-28T05:50:00Z”
kso:SO9991	sun:groups	1
kso:SO9991	sun:spots	12
kso:SO9991	sun:site	kso:Site
kso:SO9991	sun:observedBy	kso:Scientist18

Table 1. Observation data on Node A.

kso:Scientist18	a	p:Person
kso:Scientist18	p:firstName	”Sepp”
kso:Scientist18	p:lastName	”Falcon”
kso:Scientist3	a	p:Person
kso:Scientist3	p:lastName	”Klug”
kso:Scientist3	p:email	”kl@me.cc”

Table 2. Site and scientist data on Node B.

of the scientist who observed the solar phenomenon. The overall workflow of the query engine is the following:

1. grouping of triple patterns according to subjects
2. determining subject types
3. enumerate access plans
4. enumerate join plans
5. add root projection

The first step is the grouping of the basic graph pattern[1] according to the subject variables `?so` and `?obs`:

g_1	`?so a sun:SunspotObservation .`
	`?so t:inCalendarClockDataType ?dt .`
	`?so sun:spots ?sp .`
	`?so sun:groups ?gr .`
	`?so sun:observedAt ?si .`
	`?so sun:observedBy ?obs .`

g_2	`?obs p:lastName ?ln .` `OPTIONAL { ?obs p:email ?em } .`

In a second step the classes of all involved subjects are determined using the `rdf:type` property (resp. the `a` shortcut) and the catalog. Because there is a qualified cardinality restriction on `sun:observedBy`:

```
sun:SunspotObservation rdfs:subClassOf [
   a owl:Restriction ;
   owl:onProperty sun:observedBy ;
   owl:allValuesFrom p:Person ] .
```

it can be deduced, that the subject in g_2 must be a `p:Person`. Thus, the subject types for the graph pattern groups are $g_1 \rightarrow$ `sun:SunspotObservation` and $g_2 \rightarrow$ `p:Person`. It has to be mentioned that the mediator will reject any query missing type information since this is essential for data source selection. For instance, a query with a graph pattern like $\{$`:s ?p ?o.`$\}$ cannot be executed at the moment. The next step is calling the dynamic programming algorithm and enumerating all possible access plans for each graph group. For each data source providing sunspot observations (Node A, Table 1) respectively person data (Node B, Table 2), an additional access plan will be generated. If a wrapper provides different access plans using different access methods, multiple plans may be generated but all expensive plans are pruned and the cheapest one will be selected. For the given query, two access plans are generated:

$R\_Fetch(g_1, f_1, \text{”}NodeA\text{”})$
$R\_Fetch(g_2, null, \text{”}NodeB\text{”})$
f_1 = `"FILTER (?dt > \"2006-07-26T06:00:00Z\"^^xsd :dateTime && ?dt <= \"2006-07-27T06:05:00Z\"^^xsd :dateTime)"`.

Each $plan\_access$ operator (e.g. R_Fetch) returns an RDF relation with the heading variables of the corresponding group g as a sub result. For instance, the RDF relation obtained for $R\_Fetch(g_2, null, \text{”}NodeB\text{”})$ has the header `(?obs, ?ln, ?em)` and contains all corresponding tuples stored on Node B including instance URIs for `?obs`. After generating all possible join plans according to capabilities of wrappers and provided enumeration rules and selecting the cheapest plan, the root projection $p =$ `(?dt, ?sp, ?gr, ?ln, ?em)` is added and the query plan is complete:

$$M\_Proj(p, M\_Join(R\_Fetch(g_1, f_1, \text{”}NodeA\text{”}), R\_Fetch(g_1, null, \text{”}NodeB\text{”}), \text{”}?obs\text{”}))$$

[1] Named graph patterns are currently not supported.

```
PREFIX sun: <http://gsdam.sf.net/global/science/
    astro/solar#>
PREFIX p: <http://daml.umbc.edu/ontologies/
    ittalks/person#>
PREFIX t: <http://www.isi.edu/~pan/damltime/time-
    entry.owl#>
SELECT ?dt ?sp ?gr ?ln ?em WHERE {
   ?so a sun:SunspotObservation ;
      t:inCalendarClockDataType ?dt ;
      sun:spots ?sp;
      sun:groups ?gr;
      sun:observedBy ?obs .
   ?obs p:lastName ?ln .
   OPTIONAL { ?obs p:email ?em } .
   FILTER (?dt > "2006-07-26T06:00:00Z"^^xsd:
       dateTime && ?dt <= "2006-07-27T06:05:00Z"
       ^^xsd:dateTime) }
```

Listing 1. SPARQL sample query.

The sub-results from Node A and B are joined at the mediator by variable j ="?obs" and the overall result is stored in the cache attached to the mediator. Finally, the client will receive a reference to the result row counter to iterate over the results:

?dt	?sp	?gr	?ln	?em
2006-07-26T06:00:00Z	13	1	Falcon	
2006-07-27T05:49:00Z	16	2	Klug	kl@me.cc
...				

5. Conclusion

Ontologies based on RDF enable the description of data by commonly shared semantics. Because ontologies can be organized in different domains by means of namespaces, they are well suited for definition of global schemes in data integration systems. The requirements of a middleware for semantic integration of distributed heterogeneous data sources on the Grid can be met by the proposed mediator-wrapper approach. However, the framework described in this paper is a first outlook of work in progress and performance tests, extensions and improvements concerning optimization will be required to prove the approach in a production environment. Currently there is hardly any reasearch done concerning distributed query processing for SPARQL. The proposed client/server protocol as part of the SPARQL recommendation does not include any iterator-based concept. However, for efficient distributed query processing pipelining will be necessary.

References

[1] 3store. http://threestore.sourceforge.net/. Last visit: March 07.

[2] Austrian Grid. http://www.austriangrid.at. Last visit: Feb 2007.

[3] OpenLink Virtuoso. http://www.openlinksw.com/virtuoso/. Last visit: March 07.

[4] Bastian Quilitz. DARQ – Federated Queries with SPARQL. http://darq.sourceforge.net/, 2006.

[5] M. Blöchl, A. Langegger, and W. Wöß. Registration of Heterogeneous Data Sources in the Case of the Grid Semantic Data Access Middleware (G-SDAM). In *Proceedings of the Austrian Grid Symposium (AGS'06)*. OCG, 2006.

[6] P. Brezany, J. Hofer, A. Tjoa, and A. Woehrer. GridMiner: An Infrastructure for Data Mining on Computational Grids. In *APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch*, 2003.

[7] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the Semantic Web Recommendations. In *Proceedings of the International World Wide Web Conference*, page 74. Hewlett Packard Labs, 2004.

[8] Chris Bizer and Richard Cyganiak. D2R Server – Publishing Relational Databases on the Semantic Web. In *5th International Semantic Web Conference*, 2006.

[9] R. Cyganiak. A relational algebra for SPARQL. Technical Report HPL-2005-170, HP Labs, Bristol, UK, 2005.

[10] Donald Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.

[11] Donald Kossmann and Konrad Stocker. Iterative dynamic programming: a new class of query optimization algorithms. *ACM Trans. Database Syst.*, 25(1):43–82, 2000.

[12] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[13] Franz Inc. AllegroGraph 64-bit RDF Store. http://www.franz.com/products/allegrograph/. Last visit: Feb 2007.

[14] Gio Wiederhold. Mediators in the Architecture of Future Information Systems. In *Computer*, volume 25, pages 38–49, Los Alamitos, CA, USA, 1992. IEEE Computer Society.

[15] S. Harris. SPARQL query processing with conventional relational database systems. In *Workshop on Scalable Semantic Web Knowledge Base System (SSWS 2005)*, 2005.

[16] S. Harris and N. Gibbins. 3store: Efficient Bulk RDF Storage. In *Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*, Oct 2003.

[17] U. HP Labs, Bristol. Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net/. Last visit: March 07.

[18] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF. In *International Semantic Web Conference*, Sardinia, Italy, 2002.

[19] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing Queries Across Diverse Data Sources. In *Proceedings of the 23th International Conference on Very Large Databases*, pages 276–285, Athens, 1997. VLDB Endowment, Saratoga, Calif.

[20] M. Nedim Alpdemir, Arijit Mukherjee, Anastasios Gounaris, Norman W. Paton, Paul Watson, Alvaro Fernandes, and Jim Smith. OGSA-DQP: A Service-Based Distributed Query Processor For The Grid. In *Proceedings of the Second e-Science All Hands Meeting*, 2003.

[21] Peter Brezany, A. Min Tjoa, Helmut Wanek, and Alexander Woehrer. Mediators in the Architecture of Grid Information Systems. In *Conference on Parallel Processing and Applied Mathematics*, 2003.

[22] Sergej Melnik. *Generic Model Management: Concepts and Algorithms*. PhD thesis, University of Leipzig, 2004.

[23] The Mulgara Project. Mulgara semantic store. http://www.mulgara.org. Last visit: March 2007.

[24] Tim Berners Lee, James Hendler, and Ora Lassila. The Semantic Web – A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 2001.

[25] W3C. SPARQL Working Draft. http://www.w3.org/TR/rdf-sparql-query/. Last visit: Feb 2007.