Future Generation Computer Systems 25 (2009) 511–524

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A service-oriented system for distributed data querying and integration on Grids☆

Carmela Comito [a,*], Anastasios Gounaris [b], Rizos Sakellariou [c], Domenico Talia [a]

[a] DEIS, Universita della Calabria, Via P. Bucci, cubo 41 c, 87036 Rende, CS, Italy

[b] Department of Informatics, Aristotle University of Thessaloniki, 541 24, Greece

[c] School of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK

ARTICLE INFO

Article history:
Received 1 August 2008
Received in revised form
19 November 2008
Accepted 22 November 2008
Available online 7 December 2008

Keywords:
Data Grids
Distributed query processing
Wide-area data integeration
OGSA-DQP
XMAP

ABSTRACT

Data Grids rely on the coordinated sharing of and interaction across multiple autonomous database management systems. They provide transparent access to heterogeneous and autonomous data resources stored on Grid nodes. Data sharing tools for Grids must include both distributed query processing and data integration functionality. This paper presents the implementation of a data sharing system that (i) is tailored to data Grids, (ii) supports well established and widely spread relational DBMSs, and (iii) adopts a hybrid architecture by relying on a peer model for query reformulation to retrieve semantically equivalent expressions, and on a wrapper-mediator integration model for accessing and querying distributed data sources. The system builds upon the infrastructure provided by the OGSA-DQP distributed query processor and the XMAP query reformulation algorithm. The paper discusses the implementation methodology, and presents empirical evaluation results.

1. Introduction

The Grid is an effective infrastructure for the coordinated use and sharing of distributed resources in a dynamic manner [1], enabling the temporary pooling of resources to solve specific problems [2]. It does not refer only to resources such as CPU power, storage facilities and memory, but also to data sources.

Many scientific applications are presently facing a common challenge: managing a distributed data explosion. Detectors, medical imaging instruments, micro-arrays, and multi-sensor instruments are producing amounts of data that are rapidly exceeding the capacities of their current local data storage and computing environments. In many cases data are distributed from the very start, being produced by different research groups. Consider examples like genome and protein analysis data produced by many research laboratories in the world, biological databases containing patient data from a variety of hospitals. Moreover, experiments generating petabytes of data per year, such in radio-astronomy or in particle physics, need more data processing power than ever can be located in a single site, with data utilized by researchers all over the world. In such a context the Grid represents a suitable environment for scientific applications.

Accessing and thus sharing multiple data sources is deemed as a key point to really exploit the availability of such resources distributed over Grid nodes [2]. A Data Grid can include and provide transparent access to semantically related data resources that are maintained in different syntaxes, managed by different software systems, and are accessible through different protocols and interfaces. Data Grids that rely on the coordinated sharing of and interaction across multiple autonomous database management systems play a key role not only in scientific initiatives but in many industrial scenarios, as well. This fact has led to the production of vendor systems, such as Oracle11g and IBM DB2. At the level of Grid middleware infrastructure, two notable (and correlated) examples are the *OGSA Data Access and Integration* (OGSA-DAI) [3] and the *OGSA Distributed Query Processor* (OGSA-DQP)[1] [4–6] projects. These projects have moved toward a service-oriented architecture quite early in their lifecycle. OGSA-DAI exposes database management systems (including Oracle, MySQL, SQLServer, DB2, and so on) in a uniform way as services, whereas OGSA-DQP provides distributed query processing functionalities on top of OGSA-DAI, presenting the multiple databases as a single one. As such, OGSA-DQP can combine and integrate data from multiple data sources. To enhance performance, parallel and

☆ This work has been supported by the EU-funded CoreGrid Network of Excellence project through grant FP6-004265.

* Corresponding author. Tel.: +39 0984494756; fax: +39 0984494713.
E-mail addresses: ccomito@deis.unical.it (C. Comito), gounaria@csd.auth.gr (A. Gounaris), rizos@cs.man.ac.uk (R. Sakellariou), talia@deis.unical.it (D. Talia).

[1] OGSA-DAI/DQP are publicly available in open source form from www.ogsadai.org.uk.

0167-739X/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.11.009

adaptive query execution techniques have been investigated [7]. Nevertheless the semantic interpretation of the data rests with the user; furthermore, OGSA-DQP does not address any schema integration requirements. Hence, to a significant extent, the integration facilities provided by OGSA-DAI and OGSA-DQP are inadequate to meet the requirements of a number of Data Grid real cases.

Particularly, due to variety and heterogeneity of both applications and data-related resources, one of the major objectives here is to provide higher-level services that assist users in making use of several databases within a single application such as to combine data from separate, distributed sources to produce new results. It is frequently the case that the data that applications wish to combine will have been created by different organisations that will often have made local, independent decisions about both the best database paradigm and design for their data. Moreover, it is also likely to happen that databases holding data on a same domain may use heterogeneous notation to refer the same concept. Even within a single organization, data from disparate sources must be integrated. In order to provide facilities for addressing requests over multiple heterogeneous data stores for this more unpredictable environment, it is not possible to leave aside from both data integration and distributed query processing mechanisms. As cited before, the biological domain is an example of a real word scenario where reconciliation of data heterogeneity over distributed sources represents a key issue to be addressed. Data integration, in its own right, is one of the most persistent problems that the database and information management community deals with despite the fact that efficient techniques and approaches to reconciliate data heterogeneity have been developed (e.g., schema mediation languages, query answering algorithms, optimisation strategies, query execution policies, and so on) [8]. The need for semantic correlation of data sources is particularly felt in Grid settings. To date, only few projects (e.g., [9,10]) actually meet the schema-integration requirements that are necessary for establishing semantic connections among heterogeneous data sources. To this end, the use of the *XMAP* query reformulation algorithm for integrating heterogeneous data sources distributed over a Grid has been proposed [11]. Its aim is to develop a decentralized network of semantically related schemas, so that the formulation of semantically equivalent distributed queries over heterogeneous data sources is enabled. XMAP employs a decentralized point-to-point mediation approach to connect different data sources based on schema mappings. For instance, if a user submits a query for authors of scientific papers of a certain kind to a certain database containing information about scientific publications, XMAP can return equivalent queries referring to similar databases, provided that the semantic mappings have been defined. However, XMAP does not provide any distributed querying functionality.

A comprehensive data sharing tool needs to include both distributed query processing and query reformulation functionality. Thus far, data sharing tools in distributed environments tend to build upon non-database management systems or immature decentralized models. The main novelty of the work described in this paper is the implementation and presentation of a data sharing tool that (i) is tailored to data Grids, (ii) supports well-established and widely spread relational DBMSs, and (iii) adopts a hybrid architecture by relying on a peer model for query reformulation for retrieving semantically equivalent expressions, and on a wrapper-mediator integration model for accessing and querying distributed data sources. The functionality offered by OGSA-DQP and XMAP is complementary and provides an efficient basis on top of which comprehensive, Grid-enabled data sharing tools can be built. Here we discuss the design and implementation of a system encapsulating and combining the two aforementioned artefacts, along with their extended functionality. The result of our work is DQP-XMAP, a unifying service-oriented middleware system for distributed query processing and query reformulation driven by semantic connections, with a view to providing more complete access and integration services for data Grids. An important feature is that, although the system incorporates OGSA-DQP and XMAP, the model it conforms to is generic and enables the usage of any query reformulation or distributed query processing subsystem. A preliminary description of the system and some performance results have been given in [12].

The prototype that has been developed realizes the Grid vision with regards to data management in two orthogonal ways. Firstly, it enables both query reformulation and distributed query processing across heterogeneous data sources through extensions to the original work of XMAP and OGSA-DQP, respectively. Second, it constitutes an example of how independent systems that seem incompatible at first glance, exposed as services, can work together. OGSA-DQP accepts queries in a subset of OQL [13] (that is close to simple SQL) and supports relational databases, whereas XMAP was initially designed for XPath queries over XML databases. In our system, this language mismatch has been addressed through the development of a mapping between a subset of XPath over the XML representation of the relational schemas into OQL. Our approach is limited, in the sense that it does not aim to fully cover the XPath language, but it has proved sufficient for our environment.

The remainder of this paper is structured as follows: Section 2 discusses the motivation for the development of this prototype. Its main independent components, namely OGSA-DQP and XMAP are presented in Section 3 together with discussion of related work. Section 4 deals with the architecture, the design decisions and the implementation details. Section 5 presents experimental results that aim to provide useful insight into the actual behavior of the system, and the overheads incurred by the hybrid architecture. Finally, Section 6 concludes the paper.

2. Motivation and contributions

Our experience with OGSA-DQP is that, although it provides the key functionality for the execution of distributed queries transparently to users [4], its applicability is restricted because of two main reasons. First, the service oriented architecture and the SOAP-based communication protocol incur a high overhead when large data sets are transmitted. This drawback has been partly ameliorated through the development of adaptive techniques [14,15] that continuously track the optimum size of data chunks. Second, as users typically do not have enough information about the semantics of the data in the autonomous, third-party resources to which they are provided access, they find it difficult to formulate semantically correct queries that combine data from multiple sources.

Our approach to this problem, which is the topic of this paper, is to make the following option available and provide the corresponding tool. A user instead of writing a query across multiple databases has just to compose a query that refers to a single database. Then the system, through the XMAP algorithm that has been integrated, returns equivalent queries that refer to data stored to other databases, and subsequently, executes them automatically. Without such functionality a user would be forced first to understand the semantics of the data in all data sources, and then to compose a union query. Now a query to a single database is sufficient to trigger the query reformulation mechanism, which automatically constructs the semantically equivalent queries that will be then evaluated by the query engine.

To date, to the best of our knowledge, although standalone proposals to query reformulation and distributed query processing exist, there are no unified implementations enabling the aforementioned functionality.

3. Data access and integration in Grids: Background and related work

This section reviews the basics of data integration systems and discusses new issues and challenges to face in Internet-based settings.Then, before presenting the architecture of the DQP-XMAP prototype, we briefly describe the two systems it integrates, that is OGSA-DQP and XMAP.

3.1. Data integration

Information integration systems typically provide a uniform query interface to a collection of distributed and heterogeneous sources, giving users or other agents the illusion that they query a centralized and homogeneous information system. As such, they are considered as mediation systems between users and multiple data sources which can be syntactically or semantically heterogeneous while being related to the same domain. The existing mediator-based information systems (e.g., [16,17]) can be distinguished according to the type of mappings between the mediated schemas and the schemas of the sources in two approaches, the Global As View (GAV) and the Local As View (LAV). The GAV approach describes the global schemas as a view over all local schemas, whereas the LAV approach describes each local schema as a view over the global schemas.

3.1.1. Peer-to-peer data integration

As observed in several contexts, traditional centralized architectures for data integration are not the best choice for supporting semantic data integration, cooperation and coordination in highly dynamic computer networks. The rise in availability of web-based data sources has led to new challenges in data integration systems for obtaining decentralized, wide-scale sharing of semantically-related data. The P2P paradigm has recently been adopted in the database community to overcome the limitations of distributed database systems, namely the static topology and the heavy administration work, and to exploit the dissemination of data sources over the Internet. A P2P data integration system uses a decentralized semantic network of mappings between peer schemas to reformulate user queries over every peer in the network. This approach to data integration, despite a similar name, is not related to P2P networks. P2P networks deal with physical peers whereas in a P2P data integration system, peers are virtual; each peer corresponds to the schema of a structured data source.

According to these trends, several peer-to-peer data integration formalisms have been introduced [18–23]. All these systems focus on an integration approach not based on a global centralized schema. On the contrary, each data source represents an autonomous information system and information integration is achieved by establishing mappings directly among the various source schemas. Thus, a mapping can be defined between any pair of data sources. Since there is no central mediator node, such systems refer to data sources as peers and their schemas as peer schemas. A peer schema is virtual and represents the peer's view of the world. In these systems, every peer acts as both client and server, and provides part of the overall information available from a distributed environment without relying on a single global view. In fact, as in data integration, an open world assumption is made [24]. Peers do not have complete knowledge of the domain but every peer can contribute new answers to the users' query.

3.1.2. Grid data integration

Data integration on Grids has to deal with unpredictable, highly dynamic data volumes. Thus, traditional approaches to data integration, such as FDBMS [25] and the use of mediator/wrapper middleware [16], are not suitable in Grid settings. The federation approach is a rather rigid configuration where resource allocation is static and optimization cannot take advantage of evolving circumstances in the execution environment. The design of mediator/wrapper integration systems must be done globally and the coordination of mediators has to be done centrally, which is an obstacle to the exploitation of evolving characteristics of dynamic environments. As a consequence, data sources cannot change often and significantly, otherwise they may violate the mappings to the mediated schema. Further, the decentralized and distributed nature of the Grid indicates that a centralized structure that coordinates the activity of all of the nodes in the system may not be suitable, mostly because it can easily become a bottleneck. Our judgement has been that a decentralized approach can effectively exploit the available Grid resources and their dynamic allocation. For these reasons, following the P2P integration approach, the XMAP framework [26] for integrating heterogeneous XML data sources distributed over a Grid has been implemented (see next section).

To the best of our knowledge, there are only few works that provide schema integration in Grids. The most notable ones are *Hyper* [9] and *GDMS* [10]. Both systems are based on an approach similar to ours, i.e., build data integration services by extending the reference implementation of OGSA-DAI (see next section). The *Grid Data Mediation Service* (GDMS) is part of the GridMiner project [27] and uses a wrapper/mediator approach based on a global schema. GDMS presents heterogeneous, distributed data sources as one logical virtual data source in the form of an OGSA-DAI service. The main difference from our work is that it relies on the existence of a global schema, which is not necessarily a realistic assumption to make in Grids. *Hyper* is a framework that integrates relational data in P2P systems built on Grid infrastructures. As in other P2P integration systems, the integration is achieved without using any hierarchical structure for establishing mappings among the autonomous peers. In that framework, the authors use a simple relational language for expressing both the schemas and the mappings. Our integration model follows an approach that is not based on a hierarchical structure as well, however it focuses on XML data sources and is based on schema-mappings that associate paths in different schemas. Finally, semantic mapping across relational databases coupled with a Global As View approach is investigated in the context of the SASF project [28].

Semantics-enhanced distributed query processing over large-scale networks is also addressed in [29,30]. However, in such works the focus is not on semantic data integration but rather on providing a decentralized semantic overlay to support semantic search in Knowledge Grid applications. In [29] the authors propose a platform to support complex queries in a dynamic large-scale network environment. Particularly, the system supports index-based path queries by incorporating a semantic overlay with an underlying structured P2P network that provides object location and management services. Queries are forwarded along the chains of semantic object pointers to search for objects. In [30] the DST (Distributed Suffix Tree) overlay is proposed. The DST is an intermediate layer between the DHT overlay and the semantic overlay that supports search of keyword sequences by adopting the sequential semantic relationship approach between words.

3.2. The OGSA-DQP system

OGSA-DQP is an open source service-based Distributed Query Processor (DQP) supporting the evaluation of queries over collections of potentially remote data access and analysis services. The version of OGSA-DQP used in our prototype builds upon the

Globus Toolkit 4 WSRF infrastructure [31] and on top of the WSRF 2.2 OGSA-DAI release [3].

More specifically, OGSA-DQP [5,4,6] is a service-based distributed query processor. OGSA-DQP is service-based in two orthogonal dimensions:

- It manifests itself as a collection of services, and
- It accesses remote data repositories and analysis tools that are in the form of services.

OGSA-DQP services are built on top of *Data Services (DSs)* developed in the context of the OGSA-DAI project [3], which aims to provide a common service interface to Grid-connected DBMSs. DSs are simple WS-I or WSRF services and they expose multiple local *Data Service Resources (DSRs)*. One DSR may correspond to a single data resource such as an XML or a relational DBMS. However, there are special DSRs, called *factory resources* that can be used for the runtime creation of other data service resources.

The operations a data service resource can perform are called *activities*. For each activity to be called, there needs to be a separate, dedicated *activity element* in the *perform* document received by the data service resource. Activities are also the extensibility point of OGSA-DAI and DQP, i.e., additional functionalities are implemented as new activities. The two main activities that characterize the prototype described in this paper cover the execution of declarative queries (which is a basic OGSA-DQP functionality) and the identification of semantically similar queries (which is an additional functionality developed in the context of this work).

OGSA-DQP provides two types of services, *Grid Distributed Query Services (GDQSs)* and *Query Evaluation Services (QESs)*. The former are visible to end users, accept queries from them, construct and optimise the corresponding query plans and coordinate the query execution. GDQSs encapsulate and reuse the Polar* compiler for query compilation and plan generation [32]. Query evaluation services are hidden from the users, implement the query engine, interact with other services (such as OGSA-DAI services, ordinary Web Services and other evaluators), and are responsible for the execution of the query plans created by a GDQS. The interactions and functionality of OGSA-DQP services are described in detail in [5].

Some of the most notable characteristics of the OGSA-DQP service are that they are lightweight, and communication between service instances occurs in the form of XML documents transmitted over SOAP/HTTP. Moreover, QESs can evaluate *operation_calls*, which constitute calls to WSs, although they are conceived by the compiler as typed *user-defined functions (UDFs)*. Finally, note that OGSA-DQP, same as OGSA-DAI, adopts the factory paradigm: GDQS is actually an extended type of a factory resource, which enables the runtime creation of DQP DSRs for each different configuration, i.e., for each different combination of databases, physical machines hosting query evaluators and WSs playing the role of UDFs.

3.2.1. Query planning and evaluation in OGSA-DQP

Query planning in OGSA-DQP follows the two-phase optimization approach, according to which, firstly, a non-distributed query plan is generated, and secondly, different segments of that plan are allocated to different query evaluation nodes. The query planning phase is followed by query execution, whereas no changes in the query plan are allowed on the fly in the publicly available version of OGSA-DQP. The planning and execution phases are described in more detail next.

3.2.1.1. Query plan generation. The single-node, non-distributed plan is constructed by applying logical and physical optimization to the initial output of the parser of the query expression. The logical optimizer operates as follows.

- It *normalises* the plan in monoid calculus [33], which involves (i) query unnesting, (ii) fusion of multiple selection operators into a single one, and (iii) application of DeMorgan's laws to the predicates.
- It maps the monoid calculus into the logical algebra of [33]. The logical algebra contains object-relational operators such as *scan, unnest*, and *project* without specifying their actual implementation, when more than one exists (e.g., *hash join* and *nested loop* are both possible implementations of the join logical operator).
- It creates multiple equivalent logical plans and chooses the one that results in the production of less intermediate data. Changing the order of the operators and the shape of the query results in plans that produce different numbers of intermediate results.
- It pushes, or inserts, projections as close to the scans as possible.

The multi-node optimizer comprises two components: the *partitioner*, which splits the plan into fragments, thereby defining the points where communication over the network takes place, and the *scheduler*, which allocates a set of machines to each such fragment. The first step to transform a single-node plan into a multi-node one is by inserting parallelisation operators into the query plan, by using the *exchange* operator [34]. Exchanges consist of two parts that can run independently: exchange producers and exchange consumers. The producers have an outgoing buffer for each consumer, in which they add the tuples they collect from the operators upstream in the query plan. For each *exchange* operator, a data distribution policy needs to defined, in order to identify the consumer for each tuple. The policies supported include *round_robin*, *hash_distribution* and *range_partitioning*. The last two policies provide support for non-uniform data distribution among instances of the same physical operator, something which is desirable in heterogeneous environments. The fact that data is shipped in blocks rather than in individual tuples, greatly helps in reducing the communication cost incurred. Moreover, data transmission takes place concurrently with processing of the rest of the operators in the query plan. In this way, the computation and the communication costs overlap resulting in decreased query total response time. Note that mitigating the impact of high data transmission in service based systems is an ongoing activity already yielding significant results (see [14,15,35]) from which our unifying architectural proposal can benefit.

The partitioner firstly identifies whether an operator requires its input data to be partitioned by a specific attribute when executed on multiple processors (for example, so that the potentially matching tuples from the operands of a join can be compared). These operators are called *attribute sensitive* operators [36]. Secondly, the partitioner checks whether data repartitioning is required, i.e., whether data needs to be exchanged among the processors, for example for joining or for submitting to an *operation_call* on a specific machine. The final phase of query optimisation is to allocate machine resources to each of the subplans derived by the partitioner, a task carried out by the scheduler.

3.2.1.2. Query evaluation. The query engine implements each operator according to the *iterator* model [37]. As such, operators implement three main methods: *open*, *next*, and *close*. The root operator of the query tree calls *open* on its children, the children call it on their children, and so on until the *open* call is propagated to the leaf operators. *Next* is called in the same manner for each tuple, until there are no other tuples to be processed. At that point,

the propagation of *close* occurs to finish the execution in a tidied-up way. In centralised query processing, applying the *iterator model* results in a pure pull-based mode of execution. The presence of *exchanges* alters this characteristic as it enforces its producers and consumers to run independently in different threads. Thus, in a multi-node setting, parts of the query plan are executed simultaneously.

In summary, query execution in OGSA-DQP is performed as follows. The evaluators receive a plan fragment from the query compiler, which is passed on to them via their WSDL interface. Next, the operators in the plan fragment are executed as iterators. This operator execution may in turn lead to the transmission of data between nodes, using SOAP over HTTP. The steps for setting up distributed query configurations, submitting and executing queries are as follows:

(1) A GDQS factory service is deployed on a host machine that plays the role of query coordinator.
(2) Through a client interface, the users configure the DQP setting and create a specific GDQS service resource by specifying the machines, databases and WSs they want to integrate.
(3) The coordinator creates a global database schema by collating the schemata of the databases specified during the previous phase; only naming conflicts are resolved at this stage. The WSs are interpreted as typed UDFs, based on their WSDLs. After this step, the GDQS service resource created is ready to accept queries. Note that this resource is permanent.
(4) The client submits a query to the query coordinator specifying the GDQS service resource it refers to. The latter is responsible for parsing the query statement, creating a query plan, optimizing and parallelising it.
(5) The static coordinator dynamically chooses the QESs needed for query evaluation, enacts query execution and collects the query results.
(6) The results are returned to the client.

Steps (4)–(6) can be repeated any number of times for a resource created in steps (1)–(3).

3.3. The XMAP framework

The XMAP framework [11] is a decentralized network of semantically related schemas that enables the formulation of queries over heterogeneous, distributed data sources. XMAP abstracts from the underlying network infrastructure; it is modeled as a number of autonomous nodes (that can be also referred to as sites, sources, peers, etc.) holding information, which are linked to other nodes by mappings. More precisely, XMAP is composed of a collection $\mathcal{N}$ of *nodes* which are logically bound to XML data sources. That is, each data source D_n is represented by exactly one node n and, conversely, each node has access to a single data source, named *local data source*. Naturally, a *local schema* S_n is associated to this data source D_n. Data sources employ the XML data model and each source defines its own XML Schema. Each node also holds a collection of mappings M_n from its local schema to other foreign schemas. Finally, a node knows a list (also named *partial view* or, simply, *view*) of other nodes (called *neighbors*). These nodes are connected to each other through declarative mappings rules. Thanks to such mappings it is possible to translate queries from a source to another.

We recover schema heterogeneity by mapping different schemas following the peer-to-peer (P2P) integration approach recently adopted in the database community and described in Section 3.1. This approach is not based on a global schema but each database (peer) represents an autonomous information system, and data integration is achieved by establishing mappings directly among the various peers. Such an approach relies on pre-existing centralized query reformulation techniques also cited in 3.1.

3.3.1. Schema-mappings

Our approach is based on schema mappings to translate queries between different schemas. The goal of a schema mapping is to capture structural as well as terminological correspondences between schemas. To perform translation, we must understand how two schemas correspond to each other. We use a simple form of correspondence: element (attribute) correspondences that also specify the logical access paths that define the associations between elements involved. Intuitively, an element correspondence is a pair of a source element and a target element. We require the database administrator or the final user to supply only very simple correspondences. These correspondences can either be created by hand or through some (semi-)automatic mapping discovery algorithm. From such correspondences we specify mappings as path expressions that relate a specific element or attribute (together with its path) in the source schema to related elements or attributes in the destination schema.

While semantically simple, we use simple element correspondences for two main reasons. First of all, our mappings are to be considered in a large-scale framework and we do not expect a database administrator/user to know the rule machinery, whereas it is reasonable to assume that even users unfamiliar with the complex structure of the schema can provide such correspondences. Therefore, differently from related works, XMAP does not require heavyweight mapping creation from the user who has only to establish simple correspondences among paths in different schemas and the system supplies the rest. In particular, XMAP uses an algorithm which automatically determine rewritings of the user query, from the correspondences. Hence, the design was motivated by practical considerations. In addition, automated techniques for schema matching (including CUPID, LSD and DIKE) have proven to be very successful in extracting such correspondences. Moreover, note that our focus is on middleware for the Grid, therefore we do not really compete with most of the P2P data integration works that are mainly targeted towards theoretical issues.

The data integration model we propose is indeed based on path-to-path mappings expressed in the XPath [38] query language, assuming XML Schema as the data model for XML sources. Specifically, this means that a path in a source is described in terms of XPath expressions. As a first step, we consider only a subset of the full XPath language. The expressions of such a fragment of XPath are given by the following grammar:

$$q \rightarrow n|.|q\ /\ q \mid q\ //\ q \mid q\ [\ q\ ]$$

where "n" is any label (node tests), "." denotes the "current node", "/" indicates the child axis (/) whereas "//" the descendant axis, and "[ ]" denotes a predicate.

A schema mapping is defined as a set of "formulas" that relate a pair of schemas. More precisely, we define a mapping M over a source schema S as a set of mapping rules $\mathcal{R}^M = \{R_1^M, R_2^M, \ldots R_k^M\}$. As we perform path-to-path mappings, a mapping rule associates paths in different schemas. Specifically, a mapping rule is an expression of the form:

$$R^M : \{S_S, P_S\} \longrightarrow_{C^M} \{S_D, P_D\},\ \text{where:}$$

R^M is the label of the rule; S_S is the source schema with respect to which the rules are established; P_S is a path expression in the source schema; S_D is the target schema with respect to which the semantic connections are established; P_D is a path expression in the destination schema (the cardinality of this element may be more than one); C^M is the element denoting the cardinality of the mappings between the two schemas. Mappings are classified as 1-1, 1-N, N-1, N-N according to the number of nodes (both elements and attributes) of the schemas involved in the mapping relationship.

```
<schema targetNamespace="http://XMAP/XMAPDocument"
        xmlns="http://www.w3.org/2001/XMLSchema" ?>
 <element name="Mapping">
  <complexType>
   <sequence>
    <element name="sourceSchema" type="string"
             minOccurs="1" maxOccurs="1"/>
    <element name="Rule" minOccurs="1">
     <complexType>
      <sequence>
       <attribute name="Cardinality" type="string"
                  minOccurs="1" maxOccurs="1"/>
       <element name="sourcePath" type="string" minOccurs="1"/>
       <element name="destSchema" type="string"
                minOccurs="1" maxOccurs="1"/>
       <element name="destPath" type="string" minOccurs="1"/>
      </sequence>
     </complexType>
    </element>
   </sequence>
  </complexType>
 </element>
</schema>
```

Fig. 1. XML schema for XMAP documents.

The mapping rules are specified in XML documents called XMAP documents. Each source schema in the framework is associated to an XMAP document containing all the mapping rules related to it.

The structure of XMAP documents is conform to the schema shown in Fig. 1. One can notice the presence of a single `sourceSchema` element, and a set of `Rule` elements defining the mapping rules. `Rule` elements have a complex structure which specifies the paths involved in the mappings and the cardinality constraints among them.

3.3.2. Query reformulation in XMAP

Our query processing approach exploits the semantic connections established in the system by performing the *XPath query reformulation algorithm* before executing the input query, in order to gain further knowledge. This way, when a query is posed over the schema of a source, the system will be able to use data from any source that is transitively connected by semantic mappings. Indeed, it will reformulate the given query expanding and translating it into appropriate queries for each semantically related source. Every time the reformulation reaches a node that stores no redundant data, the appropriate query is posed on that node, and additional answers may be found. Thus, a user can retrieve data from all the related sources in the system by simply submitting a single XPath query.

The rationale of the algorithm is to perform individual reformulation steps. A reformulation step corresponds to the reformulation of a given query with respect to the schemas directly connected to it only. Therefore, the algorithm is composed of several reformulation steps, and each of such steps performs direct reformulations by using the point-to-point mappings. To obtain transitive reformulations of a query it is necessary to concatenate individual reformulation steps. Each time a reformulated query is obtained, the algorithm tries to rewrite it by recursively invoking the XMAP algorithm using its direct mappings.

The query reformulation algorithm uses as input an XPath query Q formulated over the schema S and the mappings M concerning S, and it produces as output zero, one or more reformulated queries Q_{R_i}.

We performed an extensive set of experiments to evaluate the performance and effectiveness of our approach. From such experiments we can realize that XMAP addresses the scalability concern scaling well with the number of participating nodes and guaranteeing quick production of reformulations, within few milliseconds even for the most demanding configurations [39].

In our architecture the reformulation algorithm has been re-engineered as a Web Service, referred to as *XMAPAlgorithm Web Service (XMAP-WS)*.

4. System architecture

4.1. The hybrid model

A comprehensive data integration architecture needs to combine both the query reformulation and the query processing services. Motivated by this reason, we extended the OGSA-DQP distributed query processor by introducing on top of it a data integration service based on the reformulation of the original query. The proposed system adopts a wrapper/mediator-based approach to integrate data sources, and it is characterized by three core components: a *query reformulator engine*, a *distributed query processor*, and a *wrapper module* (see Fig. 2).

XMAP plays the role of the reformulation engine by recovering semantic heterogeneity over data sources, OGSA-DQP is the distributed query processor with GDQS playing the role of mediator, whereas wrappers are provided by OGSA-DAI. At high level, the behavior of the developed system can be briefly outlined as follows. The user query is handled by the reformulator engine that through the XMAP query reformulation algorithm produces zero, one or more reformulations of the original query. All the obtained reformulations (including the original query) are then processed by the DQP module that partitions each of such queries in several sub-queries to be executed in parallel. Then, each produced sub-query execution plan is processed by OGSA-DQP's Query Evaluation Services (QESs) that access data sources through OGSA-DAI data service resources and produce the query result.

The model above can be implemented in several ways. Three main options include: (i) to incorporate the reformulation algorithm within the DQP module; (ii) to map the reformulation algorithm to a stand-alone module that is called from within DQP; and (iii) to make the reformulation algorithm a stand-alone module that is called from a third module that acts as the bridge between DQP and the former module. The basic advantage of the first option is that the overheads due to inter-module communication are minimized at the expense of generality. In the third option, both DQP and query reformulation algorithms can be replaced and modified easily, i.e., the design is more generic, but the system becomes less efficient. In our system, we followed the second option that is the middle solution, as shown in Fig. 3. In our architecture, the query reformulation algorithm is exposed as a completely independent service, so that any such algorithm can be plugged in and out of the system. However, this service is called from within OGSA-DQP, which has been extended accordingly, and as such, if we want to replace OGSA-DQP with another DQP system, we need to re-implement these extensions.

After having exposed the XMAP framework as a stand-alone WS, reusing it for other examples and in other settings does not pose significant problems. However, in order to integrate it with OGSA-DQP as in our case, two main technical issues had to be overcome. The first was to adhere to OGSA-DAI design principles, which entailed that the additional data integration functionality must be implemented in the form of an OGSA-DAI activity with all its complexities. The second main technical challenge stemmed from the fact that a solution to the language mismatch problem had to be developed. Both issues are discussed in the next subsection.

4.2. System functionalities

The main features, that provide added value to stand-alone OGSA-DQP are *query reformulation* and *query transformation* that we have integrated in OGSA-DQP by modifying the DQP

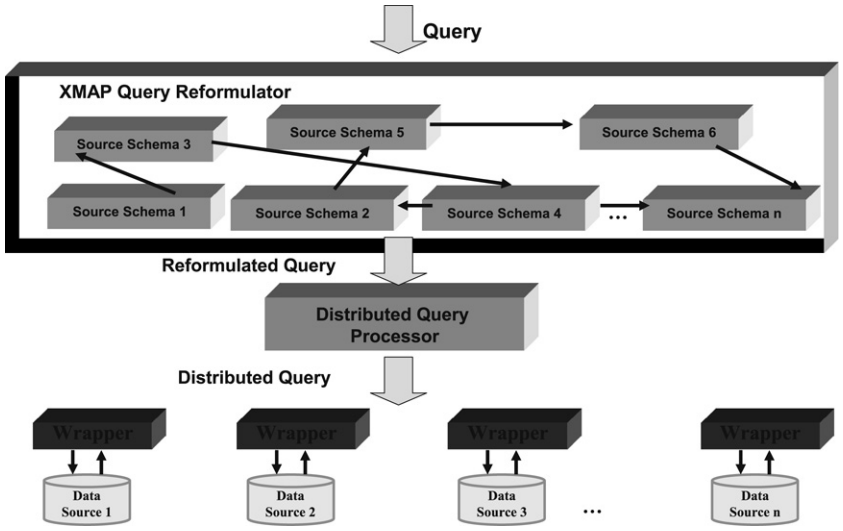

Fig. 2. System model.

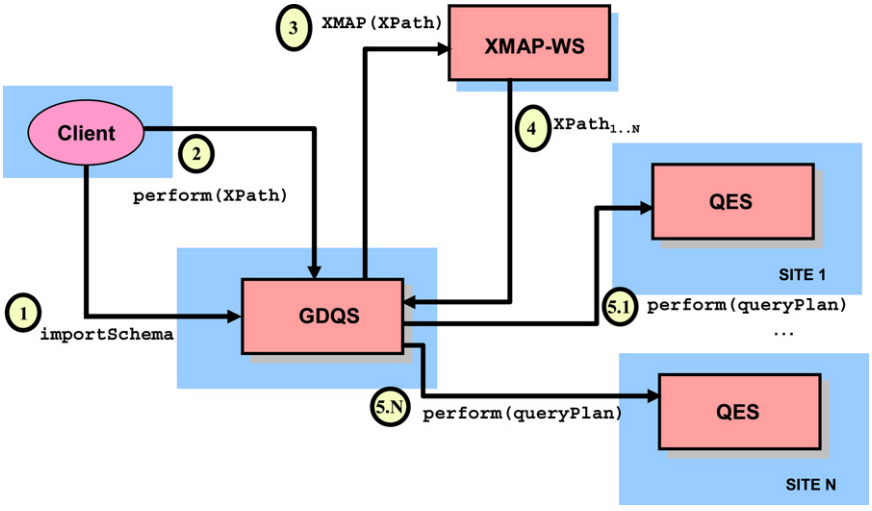

Fig. 3. The high-level architecture. The arrows denote (remote) service calls.

coordinator to: (i) locate the XMAP-WS forwarding it the XPath query; (ii) wait until the XMAP-WS produces a set of semantically equivalent queries; (iii) call a module that given an XPath query it produces the equivalent OQL one. To this aim we have implemented a new activity, called *XPathMappingActivity*. The schema of the new activity is shown in Fig. 4. The *expression* element contains the submitted XPath query, whereas *ServiceLocation* contains the address of the web service to be contacted for the actual query reformulation, according to the architecture discussed previously. As such, each OGSA-DQP data service resource supports two main operations, one for OQL queries and one for XPath queries that are reformulated and translated into OQL (see Fig. 5).

As mentioned before, query reformulation is enabled with the help of the XMAP-WS, the descriptor of which is shown in Fig. 6. However, the reformulated queries returned by the XMAP-WS cannot be evaluated in their current form by OGSA-DQP, as the latter accepts only OQL queries. Thus a query transformation step is required. The policy for that is as follows. In general, the set of meaningful XPath queries over the XML representation of the schema of relational databases supported by OGSA-DQP fits into the following template:

$/database\_A[predicate\_A]/table\_A[predicate\_B]/column\_A$

where

$predicate\_A ::= table\_pred\_A[column\_pred\_A = value\_pred\_A]$,

and

$predicate\_B ::= column\_pred\_B = value\_pred\_B$

As such, the mapping to the `select, from, where` clauses of OQL is straightforward. *column_A* defines the `select` attribute, whereas *table_A*, *table_pred_A* populate the `from` clause. If *column_pred_A* = *value_pred_A*, *column_pred_B* = *value_pred_B* exist, they go into the `where` field.

The approach above is simple but effective; nevertheless two important observations are: firstly, it does not benefit from the full expressiveness of the XPath queries supported by the XMAP framework, and secondly, although it allows for join queries, it requires the join conditions between tables *table_A*, *table_pred_A* to be inserted in a post-processing step. Such a transformation falls in an active area of research (e.g., [40,41]), and is implemented as an additional component within the query compiler. Even though, the transformation into OQL does not limit the scalability of the system that it is inherently scalable due to the P2P nature of XMAP and the OGSA-DQP's support for parallelism. This does not mean that there is not scope for further improvement. As an example, Section 3.2.1 outlines how issues related to data transmission impact on the overall system performance and scalability.

```
<?xml version="1.0" encoding="UTF-8"?> ...
 <xsd:schema
 <xsd:complexType name="XPathMappingType">
  <xsd:complexContent>
   <xsd:extension base="gds:ActivityType">
     <xsd:sequence>
       <xsd:element name="expression"
            minOccurs="1" maxOccurs="1">
         <xsd:complexType mixed="true">
           <xsd:complexContent>
            <xsd:extension base="gds:ActivityInputType"/>
           </xsd:complexContent>
         </xsd:complexType>
       </xsd:element>
       <xsd:element name="ServiceLocation"
            minOccurs="1" maxOccurs="1">
        <xsd:complexType mixed="true">
         <xsd:complexContent>
          <xsd:extension base="gds:ActivityInputType"/>
         </xsd:complexContent>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="webRowSetStream"
            minOccurs="1" maxOccurs="1">
        <xsd:complexType mixed="true">
         <xsd:complexContent>
          <xsd:extension base="gds:ActivityOutputType"/>
         </xsd:complexContent>
        </xsd:complexType>
       </xsd:element>
     </xsd:sequence>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
 <xsd:element name="XPathMappingStatement"
              type="gds:XPathMappingType"
              substitutionGroup="gds:activity"/>
</xsd:schema>
```

Fig. 4. The schema of the new activity.

Fig. 5. Fragment of the activity configuration document of an OGSA-DQP data service resource.

```
<?xml version="1.0" encoding="UTF-8"?> <deployment
xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <globalConfiguration>
   <parameter name="adminPassword" value="admin"/>...
 </globalConfiguration>...
 <service name="XMAPAlgorithmService" provider="java:RPC">
   <parameter name="className" value="xmap.XMAPAlgorithm"/>
   <parameter name="allowedMethods" value="*"/>
 </service>...
</deployment>
```

Fig. 6. Fragment of the WSDD of the XMAP-WS.

The interactions of the services are as follows (see also Fig. 3):

(1) The user contacts the *GDQS* through a client application and requests a view of the schema for each database he/she is interested in. The schema is returned in XML with the elements *database*, *table* and *column* forming an hierarchy. At this point, there is no assumption that a user has *a priori* knowledge of the semantics of these databases and the semantically-related ones.
(2) Based on the retrieved schema, a user composes an XPath query, which is sent to the *GDQS*, and not directly to the corresponding database service, following the OGSA-DQP approach. An example of an XPath query is "*/database*[*@dbname* = *IEEE*]/*table*[*@name* = *Publications*]/*column*[*@name* = *author*]", or, in a simpler form, */IEEE/Publications/author*.
(3) *GDQS* contacts a *XMAP-WS* service, which encapsulates the XMAP algorithm.
(4) The *XMAP-WS* retrieves the locally stored mapping schema, which contains the mapping information that links the paths in the submitted query with paths referring to other databases. It returns a set of *n* queries that all return results that are semantically similar to those of the initial query.
(5) For each of the results of the previous step, the *GDQS* transforms the XPath expression in OQL, and parses, optimises and schedules it so that a query execution plan is compiled. The resulting query execution plan is sent to the corresponding *QES*, which returns the results asynchronously, after contacting the local databases via OGSA-DAI services. The final results produced by the root QES are fetched in XML using WebRowSet.

In order to make clearer the concepts introduced, in the following we describe an example of using the overall prototype.

4.3. An example

Suppose that several publishers (such as IEEE and ACM) make their databases accessible from anywhere, and when a user wants to retrieve the publications of a specific author in a certain year it is sufficient to submit a query only to one of the databases, with the system being responsible for querying the remaining databases. Or, when querying a museum database for artifacts of a specific artist, the system is able to return similar results from other museum databases as well. Supporting scenarios like that, coupled with the emergence of Grid technologies, has motivated the architecture and the system of this paper.

Considering the publisher example, we used a modified real-world data set to validate our prototype, the *DB-Research* data set that has been created for the Piazza system [22].[2] The data set is based on data available on web sites concerning research in the database field. It includes the schemas corresponding to the structure and terminology of 19 such web sites (such as DBLP, CiteSeer, ACM Digital Library, and a few university sites). On the basis of these schemas, we have defined XMAP mappings between the schemas that are semantically similar. More precisely, for each source schema we have defined mapping rules toward, on average, three other source schemas.

In OGSA-DQP, the table schemas are retrieved and exposed in the form of XML documents. We exploit such representation of database schemas in order to integrate XMAP within the OGSA-DQP. In fact, users can have a view of the XML representation of relational schemas, so instead of translating the tabular view in a XML one, they can directly query the XML representation by using the XPath query language.

Examples of semantic mappings among the databases are illustrated in Fig. 7: here, the column "title" of the table "paper" of the database "ACM" is mapped to the column "paper" of the table "proceedings" of the database "DBLP". The latter is mapped to the column "article" of the "journal" table of "IEEE" data source. This information resides in the mapping document associated with the XMAP Web Service. Note that it is not necessary to map the ACM to the IEEE database directly. The XMAP mappings need to capture the semantic relationships between the data fields in different databases, including the primary and foreign keys. This can be done as shown in Figs. 8 and 9.

Suppose a user wants to find the *title* of the *paper* published in the *year* 2000. To this aim the following tasks are performed:

[2] Permission to use this data set has been kindly granted by Igor Tatarinov.