The Design and Implementation of OGSA-DQP: A Service-Based Distributed Query Processor

Steven Lynden [a] Arijit Mukherjee [b] Alastair C. Hume [c]
Alvaro A.A. Fernandes [a] Norman W. Paton [a] Rizos Sakellariou [a]
Paul Watson [b]

[a] *School of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK.*

[b] *School of Computer Science, University of Newcastle-upon-Tyne, Newcastle-upon-Tyne NE1 7RU, UK.*

[c] *EPCC, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, UK.*

Abstract

Service-based approaches are rising to prominence because of their potential to meet the requirements for distributed application development in e-business and e-science. The emergence of a service-oriented view of hardware and software resources raises the question as to how database management systems and technologies can best be deployed or adapted for use in such an environment. This paper explores one aspect of service-based computing and data management, viz., how to integrate query processing technology with a service-based architecture suitable for a Grid environment. The paper addresses this by describing in detail the design and implementation of a service-based distributed query processor. The query processor is service-based in two orthogonal senses: firstly, it supports querying over data storage and analysis resources that are made available as services, and, secondly, its internal architecture factors out as services the functionalities related to the construction and execution of distributed query plans. The resulting system both provides a declarative approach to service orchestration, and demonstrates how query processing can benefit from a service-based architecture. As well as describing and motivating the architecture used, the paper also describes usage scenarios, and, using a bioinformatics application, presents performance results that benchmark the system and illustrate the benefits provided by the service-based architecture.

Key words: Web Service, Data Integration, Data Grid

1 Introduction

The Grid is a software infrastructure that supports the discovery, access and use of distributed computational resources [21]. Although the Grid was originally devised principally to support scientific applications, the functionalities associated with middlewares, such as the Globus Toolkit [1], are potentially relevant to applications from many domains, in particular those with demanding, but unpredictable, computational requirements. For the most part, Grid middlewares abstract over platform or protocol-specific mechanisms for authentication, file access, data movement, application invocation, etc., and allow dynamic deployment of jobs on diverse hardware and software platforms.

In parallel with the development of Grid computing, *Web Services* (WSs) [24] are becoming widely accepted as a way of providing language and platform-independent mechanisms for describing, discovering, invoking and orchestrating collections of networked computational services. Although the stable and interoperable collection of WS standards managed by the WS-I [2] organisation covers quite modest functionalities, other standardisation activities in Oasis [3] and the W3C [4] provide comprehensive mechanisms for service description, security, management, notification and workflow description.

The principal strengths of WSs and Grid middlewares are complementary, with WSs focusing on platform-neutral description, discovery and invocation, and Grid middlewares focusing on the dynamic discovery and efficient use of distributed computational resources. This complementarity has given rise to the service-based Grids (for example the Open Grid Services Architecture (OGSA) [20]), which make the functionality of Grid middlewares available through WS interfaces.

Although the initial emphasis in Grid computing was on file-based data storage [40], the importance of structured data management to typical Grid applications is now widely recognised, and several approaches exist for developing Grid-enabled database services (e.g. [5,17]). To simplify somewhat, a Grid-enabled database service provides a service-based interface to a database as part of a wider collection of services for managing and using resources.

The provision of facilities that support application development is relevant to all service-oriented architectures. For example, in a Grid setting, applications can use Grid functionalities through toolkits [54] or Grid-enabled versions of parallel programming libraries such as MPI [19]. In the WS setting, tools

[1] http://www.globus.org
[2] http://www.ws-i.org
[3] http://www.oasis-open.org
[4] http://www.w3.org

exist to support the generation of client stubs (e.g., Axis[5]), but, more ambitiously, XML-based workflow languages have been developed to orchestrate WSs, of which BPEL4WS[6] is perhaps the most prominent. However, all of these approaches are essentially procedural in nature, and place significant responsibility on programmers to specify the most appropriate order of execution for a collection of service requests and to obtain adequate resources for the execution of computationally demanding applications.

This paper argues that distributed query processing (DQP) can provide effective declarative support for service orchestration, and describes an approach to service-based DQP on the Grid, implemented in the OGSA-DQP system, that: (i) supports queries over multiple services combining data access with analysis; and (ii) uses an infrastructure consisting of distributed services for efficient evaluation of distributed queries.

In the broad space of design options for a distributed query processor, OGSA-DQP:

(1) Supports low-cost data integration, in that we use existing OGSA-DAI wrappers to obtain access to networked resources, and in that there is no need to map source schemas to a single global model. This is consistent with the Grid ethos, in which the middleware is designed to encourage the rapid and potentially temporary deployment of integrated collections of resources.
(2) Builds on parallel database technology, in which both pipelined and partitioned parallelism are used to generate initial results early and to increase throughput. This is consistent with the Grid ethos, in which computational resources at multiple sites are acquired and combined to meet requirements as they arise.

The argument for the importance of DQP in a service-based Grid setting builds upon a claim of mutual benefit: the Grid stands to benefit from DQP, through the provision of facilities for declarative request formulation that complement existing approaches to service orchestration; and DQP stands to benefit from the Grid, due to the support provided for the discovery and allocation of computational resources, as required to support computationally demanding database operations (such as joins), and implicit parallelism for complex analyses.

The remainder of this paper is structured as follows. Section 2 describes OGSA-DAI, which provides data access capabilities in service-based Grids. Section 3 contains the principal technical contributions of the paper - a detailed description of how the OGSA-DQP engine has been realised, using ser-

[5] http://ws.apache.org/axis
[6] http://www-128.ibm.com/developerworks/library/specification/ws-bpel

vices both as architectural components in the design of the engine itself and as nodes in distributed query execution plans. Section 4 describes the range of tools that users can use to interact with the OGSA-DQP system, including a GUI client and programming toolkit. Section 5 presents the results of an experimental evaluation. Section 6 draws contrasts with other work on distributed query processing and Grid data integration. Finally, Section 7 states some conclusions. This paper reflects a number of changes to OGSA-DQP since the conference paper in which it was first reported [3]; changes of substance include a closer integration with OGSA-DAI both for query evaluation and application development; a revision to the service definitions used to support query evaluation; and the provision of an experimental evaluation.

2 OGSA-DAI

In a service-oriented Grid the principal objective is to enable computational resources to be accessed and managed in a secure and systematic manner. As databases are important computational resources, database access services can be expected to have an important place in middlewares for data Grids [6]. In this setting, the OGSA Data Access and Integration (OGSA-DAI) project [7] has developed a service-based infrastructure for accessing both relational databases and XML repositories that integrates service-based access to data resources through two WS platforms:

(1) The WS-I [8] platform in the OMII [9] middleware stack.
(2) The WSRF platform in the Globus Toolkit 4 [10] middleware stack.

The role of OGSA-DAI in a service-based Grid, illustrated in Figure 1, involves interactions between several components which are now defined:

- OGSA-DAI data service: a WS that implements various port types allowing the submission of requests and data transport operations.
- Client: an entity that submits a request to the OGSA-DAI data service; a request is in the form of a *perform* document that describes one or more activities to be carried out by the service.
- Consumer: a process, other than the client, to which an OGSA-DAI service delivers data.
- Producer: a process, other than the client, that sends data to an OGSA-DAI data service.

[7] http://www.ogsadai.org.uk
[8] http://www.ws-i.org
[9] http://www.omii.ac.uk
[10] http://www-unix.globus.org/toolkit

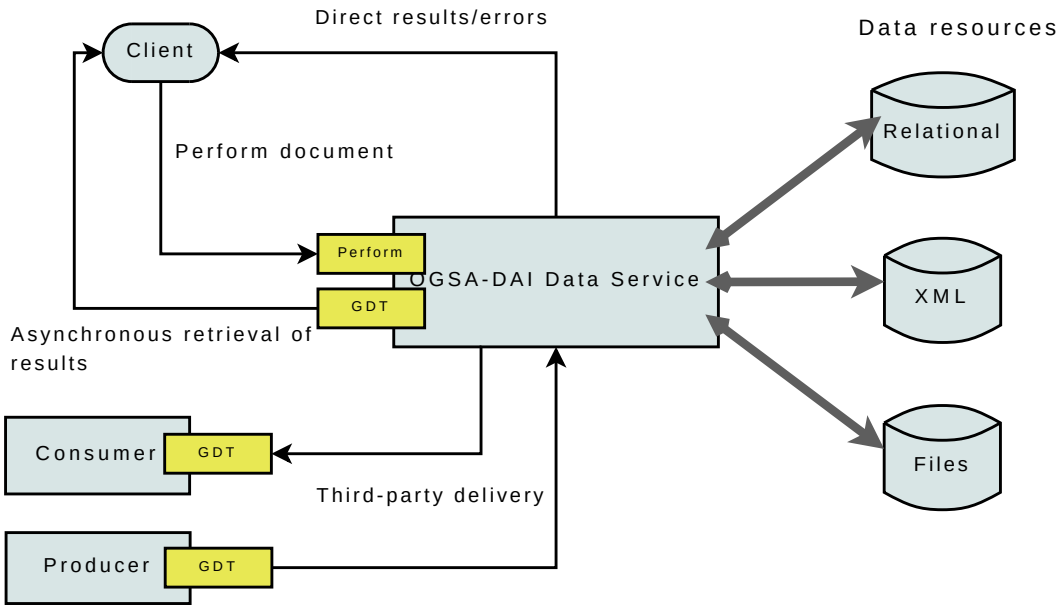

Fig. 1. OGSA-DAI Data Service

In order to make a request to an OGSA-DAI data service, the client invokes a WS operation on the data service, parameterised by a perform document. A perform document is an XML document describing the request that the client wants to be executed, defined by linking together a sequence of activities. An activity is an OGSA-DAI construct corresponding to a specific task that should be performed. The output of one activity can be linked to the input of another to perform a number of tasks in sequence. For example, the output of an `SQLQueryStatement` activity, which executes a SQL query, can be sent to a `DeliverToGDT` activity to send the results to a consumer supporting the Grid Data Transport (GDT) port type. The input of the `SQLQueryStatement` activity may be pulled from a consumer supporting the GDT port type by linking the output of a `DeliverFromGDT` activity to the `SQLQueryStatement` activity. A range of activities are supported by OGSA-DAI, falling into the broad categories of relational activities, XML activities, delivery activities, transformation activities and file activities. Furthermore, the activity is an OGSA-DAI extensibility point, allowing third-parties to define new activities and add them to the ones supported by an OGSA-DAI data service.

2.1 Data Service Resources

Data service resource is the term used within the OGSA-DAI domain for an individual resource that is managed by an OGSA-DAI data service. When a perform document is submitted to an OGSA-DAI data service, a data service resource is specified which is used by the activities contained in the perform document to execute their tasks. The use of this mechanism allows a single

service to expose multiple data resources, which can be of different types, each supporting a different set of activities. Data service resources may also expose resource properties, which are XML elements describing properties of a given resource. Resource properties may be queried and in some cases modified using the interfaces defined by the WS-Resource Properties [29] specification. OGSA-DAI data services expose resource properties describing the status of executing perform documents and the activities supported by a data service resource.

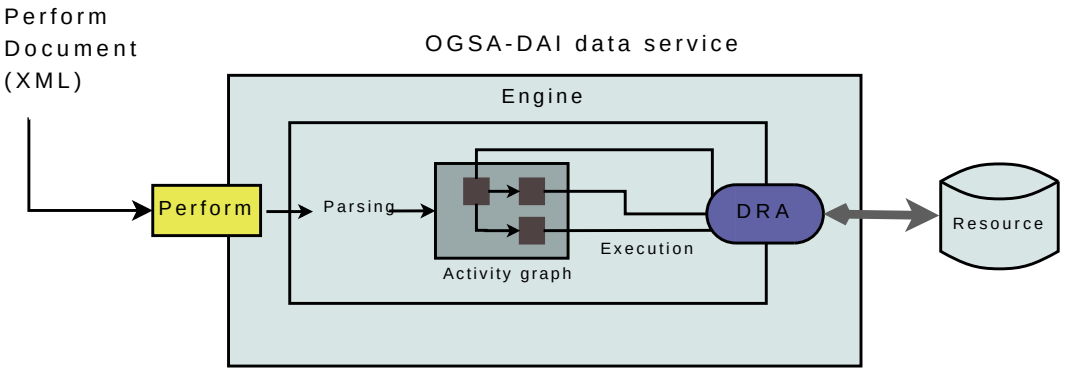

Fig. 2. Data resource accessors. When a perform document is submitted, the OGSA–DAI engine parses it and composes a directed acyclic activity graph. The engine submits each activity to a data resource accessor for execution.

Data service resources and resource properties are also OGSA-DAI extensibility points. Third parties may add their own data service resources which can extend the set of resource properties supported by OGSA-DAI. To facilitate this extensibility mechanism, OGSA-DAI introduces the *data resource accessor* (DRA). A DRA is a component that mediates communication between the OGSA-DAI engine, which processes perform documents, and the data resource on which activities are executed. Figure 2 illustrates the role played by DRAs within the OGSA-DAI data service. To support a new type of data resource, a DRA is implemented to support the execution of a set of activities on the data resource. In the next section it will be described how a data resource accessor is used to represent a federation of services over which distributed queries can be evaluated.

The service-based DQP approach described in this paper functions as an integration component allowing queries to be composed over multiple OGSA-DAI-wrapped relational data sources. Although the core OGSA-DAI data service provides a useful abstraction for *accessing* individual data resources on the Grid, they do not address challenges associated with *integrating* data from multiple resources; the following section describes how this is supported using OGSA-DQP. Thus OGSA-DAI allows a program or a perform document to interact with several different data resources, but does not support declarative query evaluation over multiple sources. Declarative query evaluation over multiple sources does not offer any additional capability compared with an

application that accesses multiple sources, but requests to a distributed query processor are likely to be considerably more concise, and benefit in the case of OGSA-DQP both from query optimization and parallel evaluation.

3 A Service-based DQP Architecture

This section describes a query processing framework, OGSA-DQP, in which query compilation, optimisation and evaluation are implemented using a service-based architecture; all of (i) the distributed query processor; (ii) the query fragment execution nodes; (iii) the data resources accessed from a query; and (iv) the computational resources invoked from queries are represented as services.

An important benefit of this approach is that OGSA-DQP can be seen to provide an effective and efficient platform for declarative orchestration of services in the Grid. As such, service-based DQP provides an alternative to procedural approaches for expressing data-based computations over the Grid.

OGSA-DQP extends OGSA-DAI with the following services:

- *DQP coordinator* service: An OGSA-DAI data service, enhanced to support distributed queries using the extensibility points discussed in the previous section. The main enhancement is the contribution of the `DQPQueryStatement` activity which compiles, optimises and schedules SQL queries for execution.
- *DQP evaluator* service: A service capable of evaluating a query fragment provided by the coordinator. An evaluator is able to play a role in the evaluation of a query by retrieving data from OGSA-DAI-wrapped data resources, invoking analysis services and managing the flow of data between other evaluators. Multiple evaluators are used, to provide the benefits of parallelism during query evaluation.

Figure 3 illustrates the architecture of OGSA-DQP. Before queries may be submitted, OGSA-DQP obtains the metadata that it needs to compile, optimise, partition and schedule distributed query execution plans over the multiple execution nodes (evaluators). When a query is submitted, the following steps take place:

(1) The client sends a perform document, containing a `DQPQueryStatement`, using the `perform` port type of the OGSA-DAI data service. The query is parsed, compiled and scheduled for execution and a query plan is produced. This query plan is partitioned, where each partition is an XML document specifying the role of an individual evaluator in the evaluation

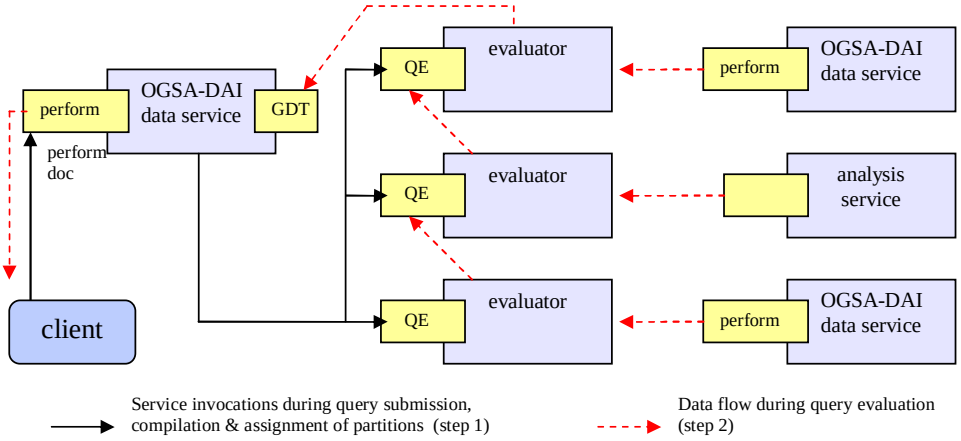

Fig. 3. OGSA-DQP Architecture

of the query. Each query partition is sent to the relevant evaluator using the query evaluation (QE) port type of the evaluators.

(2) The evaluators retrieve data from OGSA-DAI data services and invoke any analysis services required to evaluate the query. The flow of data during the execution of a query plan forms a tree, where data flows upwards via the evaluators, which use the QE port type to send data to other evaluators. The `DQPQueryStatement` activity, which is the root of the tree, is able to receive data via the GDT port type of the OGSA-DAI data service.

There are two phases involved in the use of OGSA-DQP, a set-up phase where the federation of services over which queries may be executed is composed, and a query phase during which queries are submitted and evaluated.

3.1 Setting up a distributed query service

To accomplish the set-up phase, the factory pattern is adopted and OGSA-DQP is modelled as two separate data resources, one encapsulating a factory entity and another representing a database federation over which queries may be evaluated. In addition to the activity extensibility point, another extensibility point is used that allows new types of data resources to be exposed by OGSA-DAI data services. When OGSA-DAI exposes a data resource, the engine instantiates a DRA to interface with the data resource, as was shown in Figure 2. OGSA-DQP introduces DRAs for the following two types of data resources:

- The *DQP factory* data resource, which maintains OGSA-DQP system level installation data and supports the execution of the `DQPFactory` activity, which is introduced to enable configuration.
- The *DQP* data resource, which represents a federation of data resources

and analysis services over which queries may be composed using `DQPQueryStatement` activities, and the pool of evaluator services which may be utilised to evaluate such queries.

OGSA-DAI allows data resources to be exposed dynamically by providing activities with the ability to instantiate a DRA at runtime, and this functionality is used by the `DQPFactory` activity to deploy DQP data resources. The `DQPFactory` activity has one input, which takes an XML document specifying the parameters required to configure a DQP data resource. This configuration document is provided by the client and is used to specify the data sources and analysis services that the DQP data resource should attempt to import. The output of the `DQPFactory` activity returns the resource ID (the unique, automatically assigned identifier used to identify individual resources exposed by an OGSA-DAI data service) of the dynamically exposed DQP data resource. When the activity is executed, the following sequence of events occurs:

(1) A DQP DRA is created and initialised.
(2) The `DQPFactory` activity passes the configuration document to the DRA, which attempts to import the data sources and analysis services specified. During this process physical metadata about database tables is also retrieved from the data sources. This information is used during optimisation to construct query execution plans.
(3) If schema import is successful (i.e. at least one data source is successfully imported), a DQP data resource is exposed, through the created DRA, which encapsulates the data federation over which queries can be evaluated. The unique identifier used by the OGSA-DAI data service to identify the resource is returned to the client. If schema import is unsuccessful, the DQP DRA is destroyed and an error is reported to the client.

3.2 Query submission, optimisation and evaluation

The `DQPQueryStatement` has one input, which accepts a SQL query, and one output, which supplies the result of the query in XML. The following takes place when the `DQPQueryStatement` is executed:

(1) An OGSA-DAI input stream is created. Input streams allow data to be sent to an activity, either from another activity or remotely via the GDT port type. OGSA-DAI allows input streams to be created during the initialisation of an activity, which results in the activity possessing a second input, although only the first is visible to the client. The creation of the input stream allows the `DQPQueryStatement` to function like an `InputStream` activity and receive data from remote services. Activities

such as `DeliverToGDT` and `InputStream` usually work in pairs to deliver data from one OGSA-DAI service to another, however in this case the input stream is created to receive results from evaluator services behind-the-scenes.

(2) The query is compiled to yield a partitioned query plan, where each partition is assigned to an evaluator. The compilation, optimisation and scheduling of queries is described in more detail in Section 3.3. An XML representation of the query plan is exposed as a property of the DQP data resource. This resource property allows the client to obtain a representation of the query plan as it is being executed.

(3) Each query plan partition is sent to the relevant evaluator. A query plan partition contains the relevant information (service endpoint and input stream identifier) needed by each evaluator to stream data back to the `DQPQueryStatement` activity created in step 1. Evaluators are able to stream data back to the `DQPQueryStatement` activity in the same way that a `DeliverToGDT` activity delivers data from one OGSA-DAI data service to another.

(4) The `DQPQueryStatement` activity waits for results from the evaluators. As results are received, they are converted to XML and sent to the activity's output.

(5) When the complete set of query results has been received, the input stream created in step 1 is closed and the resource property exposed in step 2 is removed.

Queries are evaluated using both *pipelined* and *partitioned* parallelism. Pipelined parallelism is achieved through the the use of a multi-threaded implementation of the iterator model [28]. In the iterator model, each operator produces data one tuple at a time, allowing produced tuples to be processed by subsequent operators in the query plan without waiting for an input operator to finish. Partitioned parallelism is facilitated by the placement by the scheduler of an individual plan partition on multiple nodes, as described further in Section 3.3. Although the iterator model operates tuple-at-a-time, inter-service requests (both between the DQP coordinator and evaluators, and between different evaluators) transmit blocks of tuples, as individual WS invocations have significant overheads.

3.3 Compiling, optimising and scheduling queries

The resources used to evaluate a query are identified by the query planner and selected for use based on the predicted needs of the queries (computed using a cost model [14]), and on the properties of the available computational resources. In practice resource assignment is principally heuristic; for example, resource assignment selects the fastest available nodes first, and prefers nodes

that are located on the same network. When a query is submitted, an execution plan is produced using the two-step optimisation paradigm that has previously been exploited for both parallel and distributed databases [35].

Phase 1: In the first phase, the query compiler performs the transformations that are valid irrespective of the number of execution nodes (parsing, followed by type checking, followed by logical and then physical optimisation) to yield a single-node execution plan. The parser supports an SQL-based syntax that is extended to allow function calls. During logical optimisation, selectivity estimates are computed based on the physical metadata obtained during schema import. The selectivity estimates are then used to create a left-deep join tree using a heuristic which aims to minimise the size of the intermediate relation produced at each stage (for details of the optimization algorithm used, see [23], Section 7.6.6). Physical optimisation simply chooses an algorithm to implement each join operator based on cost estimates for each suitable join algorithm.

Phase 2: In the second phase, a partitioner breaks down the single-node execution plan into partitions and a scheduler assigns the partitions to execution nodes. During this phase, the optimiser considers parallelising certain operators in order to speed up the query execution. There are two operator types for which parallelisation can make a significant difference to the query execution time: function calls which invoke external WSs, and specific types of hash table-based join algorithms. The optimiser considers parallelisable joins and WS calls as candidates for parallelisation and increases the degree to which they are parallelised until there are no more evaluators available or the estimated cost of further parallelising the operator outweighs the estimated benefit. This strategy is based on the approach described in [26].

The scheduler assigns query plan partitions to evaluators based on the computational characteristics of the nodes on which evaluators are deployed. The computational characteristics of a node consist of a range of attributes describing the node's physical properties which can be used to estimate the speed with which it can evaluate a given partition of the query execution plan. For example, a node with a large amount of memory may be required to implement a hash-join for which the hash table built by the join operator is expected to be large. The provision of computational metadata about execution nodes is an OGSA-DQP extensibility point, where users may provide a mechanism for dynamically updating the metadata, the implementation of which depends on the capabilities provided by the service-based Grid in which OGSA-DQP is deployed. Metadata may be obtained statically, for example using a configuration file when installing OGSA-DQP, or dynamically, for example using the Index Service [11] provided by the Globus Toolkit 4.

[11] http://www.globus.org/grid_software/monitoring/

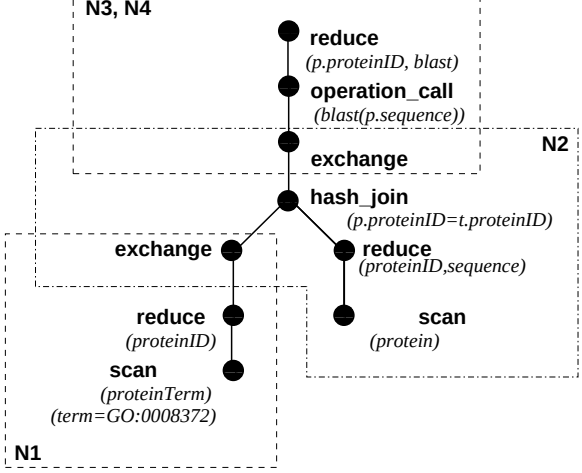

Fig. 4. Distributed Query Plan

Figure 4 depicts an execution plan produced by the compiler/optimiser. The plan has been produced for a query that joins data from two bioinformatics data sources, one of which contains protein sequences. The hash_join operator used in this plan caches the left input relation in a hash table and consumes the right input one tuple at a time, producing matching tuples via lookups using the hash table. The reduce operator is used by OGSA-DQP to implement projection, and the operation_call operator invokes Web service operations. Three partitions (i.e., the dashed regions) have been decided upon whose intersections are marked by the exchange operators [28] used to manage communication and data distribution between evaluators. Note that the partition containing the BLAST execution (producing sequence alignment scores for input protein sequences, which can be computationally expensive) has been scheduled to run on two of the four nodes N1-N4 harnessed for executing the query.

3.4 Summary

The implementation of OGSA-DQP using the OGSA-DAI extensibility points is based on two new activities, and two new data resources. As OGSA-DAI manages the concurrent execution of activities, OGSA-DQP is capable of processing multiple queries simultaneously without the need for manually developing a concurrent implementation. The use of the extensibility points allows OGSA-DQP to take advantage of the both the authentication and the host of delivery options provided by OGSA-DAI by connecting the output of `DQPQueryStatement` activities to core OGSA-DAI activities such as `DeliverToGDT` and `DeliverToFTP`. OGSA-DQP is also insulated from the multiple platform approach adopted by OGSA-DAI, which allows OGSA-DQP to be deployed using either OGSA-DAI WSRF or OGSA-DAI WS-I and therefore provides both of these options to the users of OGSA-DQP.

Other than OGSA-DAI, OGSA-DQP has few mandatory external dependencies on other Grid components or services; as such, the infrastructure can obtain information about available sevices from configuration parameters rather than by interrogating registries, and uses web service communication rather than Grid data movement protocols during query evaluation. Enhanced Grid query processors could exploit diverse data transfer or reservation capabilities, though at the cost of greater complexity in query planning and additional external dependencies for the query evaluator.

The software described in this paper is available in open-source form from *http://www.ogsadai.org.uk/dqp*.

4 Usage scenarios

Use of OGSA-DQP is facilitated by a command-line client, a GUI client, and a programming toolkit that allows users to integrate the invocation of the distributed query service with their applications. These tools provide a range of options for interacting with OGSA-DQP, depending on the requirements of the user.

4.1 OGSA-DQP clients

The functionality of the clients is divided across two distinct modes of operation: *administrator mode* and *user mode*.

The main role of the administrator mode is to set up a query session, which corresponds to the configuration of a DQP coordinator. This can be achieved (i) by passing a configuration file as an argument to the command-line client, or (ii) interactively by using the GUI client. The configuration corresponds to the set-up phase described in Section 3.1, and results in the import of schemas from OGSA-DAI-wrapped data sources and WSDL documents from analysis services. Using the GUI client, the user is able to specify the evaluators, data sources and analysis services that should be used when setting up a DQP resource, and subsequently examine the outcome of the setup phase in detail by viewing the metadata exposed by the DQP resource, describing imported schemas and available resources. Any web service that takes or returns the following types: xsd:String, xsd:Boolean, xsd:Decimal, xsd:Float, xsd:Double, xsd:Time, xsd:Date, xsd:Long, xsd:Int, xsd:Short and xsd:Byte. Figure 5 shows the GUI in administrator mode as the user browses the metadata exposed by a DQP resource.

Following successful configuration and creation of a DQP resource representing a federation of data and analysis services, both the command-line and GUI clients allow queries to be submitted and their results displayed. Here, the GUI client offers three advantages by:

- Allowing the query to be more easily composed by displaying the global schema, which consists of the union of the schemas of the sources imported from the resources being queried. The current version of OGSA-DQP does not provide support for the development of a global schema that suppresses schematic heterogeneities between sources (e.g. [34]), although OGSA-DQP has been used as a back-end evaluation engine in several projects in which schematic query evaluation has been investigated [25,55].
- Providing the user with a graphical representation of the query plan used to evaluate a query.
- Providing the facility to export results in HTML and XML formats.

Figure 6 illustrates one of the features of the GUI client during the execution of a query, where the user is examining a parallel query plan used to execute the query. Each node in the displayed query plan is an operator, which may be parallelised over a number of machines. The dotted boundaries annotated with hostnames indicate the hosts on which a group of operators have been scheduled for execution.

4.2 Programming with OGSA-DQP

OGSA-DAI provides a Java-based client toolkit for interacting with OGSA-DAI data services, which is also an extensibility point where developers can add their own client toolkit classes to support third-party data resources and activities. The client toolkit allows developers to integrate DQP with their applications using only a few lines of code. Configuration of a DQP resource can be achieved as follows:

```
1 DataService service = fetcher.getDataService(
     "http://test.man.ac.uk:8080/axis/services/dqpservice",
     "dqp-factory");
2 DQPFactory factory = new DQPFactory(config);
3 ActivityRequest request = new ActivityRequest();
4 request.add(factory);
5 Response response = service.perform(request);
```

Line 1 uses the API provided by OGSA-DAI to create a local object (`service`) representing a remote OGSA-DAI data service exposing a DQP factory data service resource. In Line 2, a `DQPFactory` object is instantiated and used to represent a DQP factory activity parameterised by `config`, which is the XML

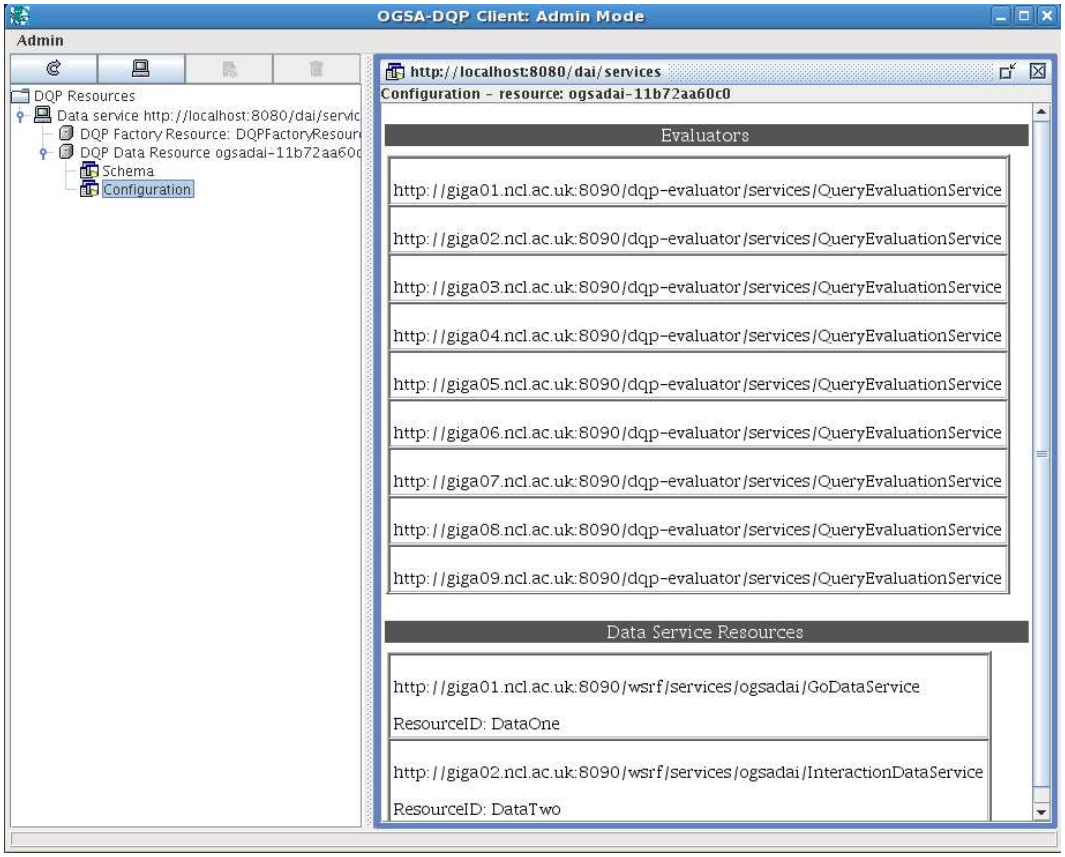

Fig. 5. OGSA-DQP GUI client in administrator mode

document used to specify configuration parameters such as the services used to form the federation over which queries may be composed. Lines 3 and 4 create an activity request (i.e. a perform document) which is sent to the data service in Line 5. The `Response` object returned in Line 5 provides methods allowing results or errors to be received from the data service. A configured DQP data service resource can be queried using the client toolkit as follows:

```
1 DataService service = fetcher.getDataService(
     "http://test.man.ac.uk:8080/axis/services/dqpservice",
     "dqp-resource");
2 ActivityRequest queryRequest = new ActivityRequest();
3 DQPQuery dqpQuery = new DQPQuery("select id from dqp_goterm;");
4 queryRequest.add( dqpQuery );
5 Response response = service.perform(queryRequest);
```

Line 1 again creates an object to represent the OGSA-DAI data service, except that this time a different data service resource is used. This corresponds to a resource created as the result of a configuration operation such as the one described above. Lines 2-4 create an activity request which contains a `DQPQueryStatement` which is encapsulated in the client toolkit by the `DQPQuery` class. Line 5 performs the request, and from the response object re-

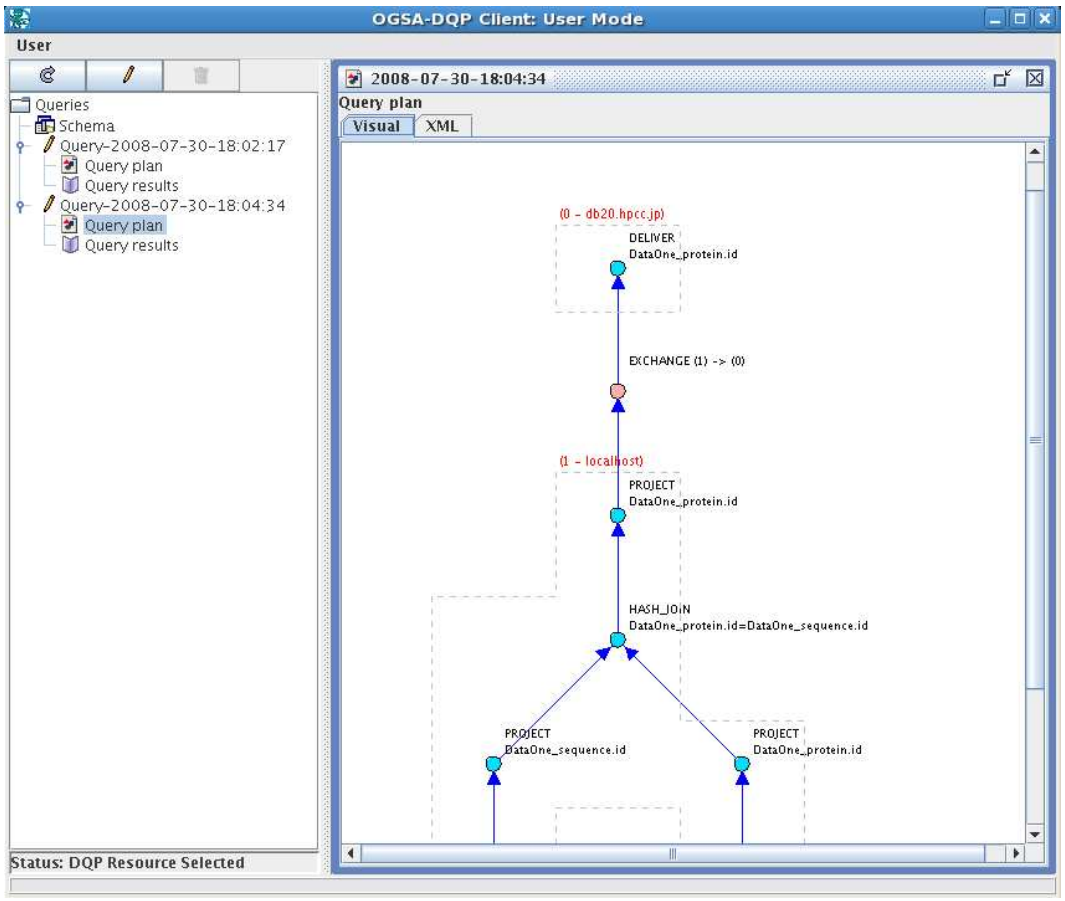

Fig. 6. Screenshot of a query plan displayed by the GUI client

sults may streamed back to the client application from the data service. The OGSA-DAI client toolkit also supports the querying of resource properties, which allows the properties exposed by DQP resources, such as the schema of the data resource federation, to be retrieved.

5 Performance Evaluation

Architecturally, OGSA-DQP differs from established distributed query evaluators in two principal ways: in using a service-based Grid to access remote data and computational resources, and in exploiting partitioned parallelism both for query-internal operations (such as joins) and for external web service calls. This section describes experiments that have been carried out that seek: (i) to illustrate the overall performance that has been obtained using a service-based distributed query processor in both local and wide-area networks; and (ii) to show the benefit of parallelism both for standard queries and for queries invoking web services that provide application-specific analyses.

```
select ORF, GOTermIdentifier
from   protein_goterm;
```

(a) SCAN-QUERY-1: scan *protein_goterm*.

```
select id, type, name
from   goterm;
```

(b) SCAN-QUERY-2: scan *goterms*.

```
select id, type, name, ORF, GOTermIdentifier
from   goterm, protein_goterm
where  goterm.id=protein_goterm.GOTermIdentifier
       and id like 'GO:001%' and ORF like 'Y%';
```

(c) JOIN-QUERY: join *goterm* and *protein_goterm*.

```
select calculateEntropy(sequence)
from   sequence;
```

(d) ANALYSIS-QUERY: compute the entropy of proteins in the sequence table.

Fig. 7. Queries used in experimental evaluation

Several research projects have reported experimental results that in some way build on OGSA-DQP, and thus which provide further evidence as to the scaleability and performance of the architecture using literature [25] and proteome [56] data resources.

5.1 Experiment Setup

The experiments involve three tables: *goterm*, which contains data from the Gene Ontology (GO) [30] on biological function; *protein*, which stores data on protein sequences; and *protein_goterm* which associates proteins with the terms that describe their functions. The *sequence* table contains the amino acid sequences of proteins belonging to multiple organisms. Figure 8 describes data volumes for each table.

As OGSA-DQP operates in a service-oriented environment, all data is shipped across the network as XML documents using SOAP as the transport protocol, which increases the volume of the data on the wire significantly.

The four queries used in the experiments are listed in Figure 7. The queries SCAN-QUERY-1 and SCAN-QUERY-2 fully scan two of the data sources mentioned above, and are used to illustrate the throughput of basic data access operations over local and remote sources. JOIN-QUERY joins the two data sources, and is used to study the effect of parallelism on join performance; a main-memory hash-join is used throughout. ANALYSIS-QUERY is used to explore the performance of parallel invocations of a WS operation. Each protein sequence is used as an input parameter to a WS call to compute its entropy (i.e. information content).

The data sources are hosted in MySQL databases. The experiments have been run on local area networks at Manchester and Newcastle universities, which

Data Source	Number of rows	Size in the database	Attributes
protein_goterm	49171	1.23MB	ORF, GoTermIdentifier
goterm	22622	1.56MB	id, type, name
sequence	8763	1.9MB	ORF, sequence

(All attributes are `String` types)

Fig. 8. The datasets used in the experiments

are connected to the UK academic network (JANET - Joint Academic NETwork) which has a 10Gb/s backbone. The machines are connected via Fast Ethernet switches (10/100Mbps) to a departmental network that consists of a fibre optic Gigabit Ethernet backbone. The computational nodes used at Manchester are AMD Athlon(tm) XP 3000+ machines, each with a 3GHz CPU and 512MB-1GB RAM. The computational nodes used at Newcastle are Intel(R) Xeon(TM) 2.80GHz CPUs with 2GB of RAM. The experiments are now described.

5.2 Experiment 1

This experiment executes table scans with increasing data volumes. The objective is to discover the time taken for OGSA-DQP to scan remote tables and to see how the response time scales as the size of the tables are increased. For this experiment, the *goterm* and *protein_goterm* tables are scanned using SCAN-QUERY-1 and SCAN-QUERY-2 respectively. This experiment is performed using three of the Manchester nodes, where two of the nodes each have (i) a database containing one of the queries tables, (ii) an OGSA-DAI data service exposing the databases, and (iii) a DQP evaluator service. Queries originate from an OGSA-DQP coordinator deployed on the third node.

5.2.1 Results

Figure 9 shows the response times of SCAN-QUERY-1 and SCAN-QUERY-2 for increasing data volumes. The two main conclusions that can be reached from this graph are that absolute response times shown are high for the amounts of data delivered and that response time scales linearly as the data volume increases. Experience reported in [32] indicates that the OGSA-DAI database wrappers are around an order of magnitude slower than JDBC calls for bulk delivery on a local area network. OGSA-DQP exploits a still-slower block-delivery interface to OGSA-DAI to allow pipelined processing, from which there is no real benefit in these simple queries; as such, there is a significant performance overhead associated with the creation and unpacking of XML data representations, and for sending in SOAP messages. Ongoing

research and development work on WS infrastructures is likely to reduce such overheads significantly (e.g. [48]).

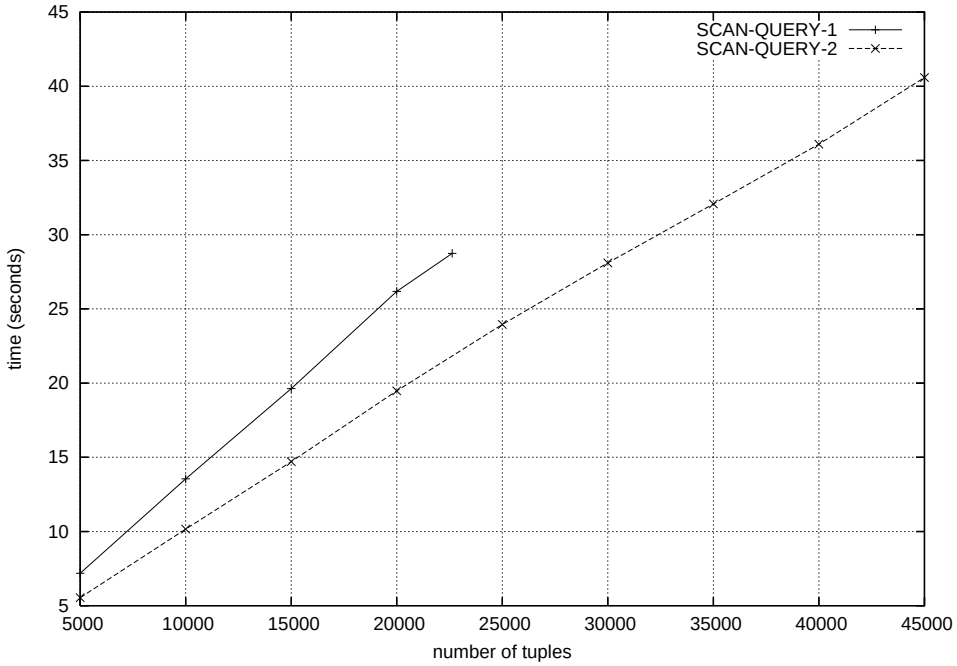

Fig. 9. Comparison of the response times for scanning two data sources

5.3 Experiment 2

This experiment aims to investigate the benefit of parallelising operation call operators that invoke WSs. ANALYSIS-QUERY is used to invoke the *calculateEntropy* operation which is made available by different WSs located on a number of separate machines. OGSA-DQP is able to parallelise the execution of analysis queries by scheduling operation call operators for execution on different evaluators, each of which invokes a separate instance of the WS operation. This experiment is performed using the machines at Manchester, where a collection of nodes each host a DQP evaluator and a WS supporting the *calculateEntropy* operation. A separate node hosts one evaluator and the *sequence* table via an OGSA-DAI data service. The number of available instances of the *calculateEntropy* operation is increased, with a total of 7 evaluators (6 evaluators on analysis service nodes and the single evaluator on the data source node) made available to the DQP coordinator for the execution of each query.

5.3.1 Results

Figure 10 shows the response times of ANALYSIS-QUERY as the number of replicated operations is increased from 1 to 6. The results show clearly that response time decreases as the level of parallelism used to execute the *calculateEntropy* operation increases. The data source scanned, along with one evaluator, reside on a node that does not host any of the WSs used to execute the query. The optimiser chooses to schedule operation-call operators to run on evaluators that are as close as possible to the WS being invoked, therefore the output of the scan operator on the data source node is always routed to a remote evaluator, resulting in some communication overhead regardless of the degree of parallelism. The results show that the communication overhead does not outweigh the benefit of increasing the degree of parallelism used to execute the operation.

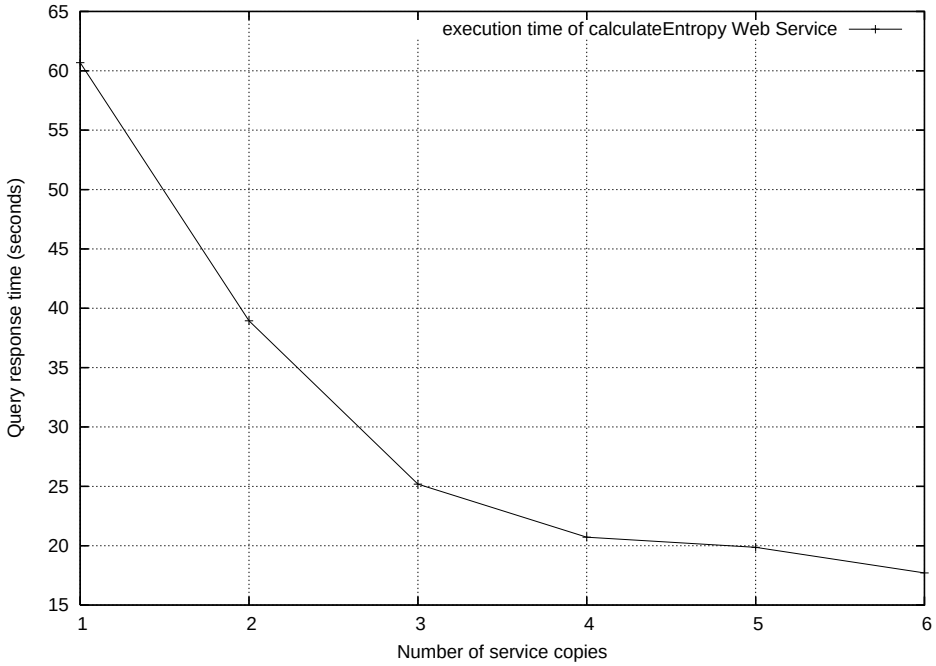

Fig. 10. Response times for analysis queries with different levels of parallelism

5.4 Experiment 3

This experiment aims to investigate the benefits of parallelising join operators by comparing the DQP framework proposed in this paper to a more centralized wrapper-mediator approach, in which a single mediator interacts with a collection of wrapped sources [33,13]. Using the wrapper-mediator architecture, data is fetched from remote sites and processed centrally using a single evaluator. In contrast, OGSA-DQP is able to process data remotely and in parallel by parallelising operators such as joins and operation calls. OGSA-DQP can

simulate the wrapper-mediator architecture if only a single local evaluator is available to the DQP coordinator, therefore necessitating the transport of all data from remote data sources to this one evaluator, which then implements the query execution plan. In this experiment, queries originate from a DQP coordinator deployed on one of the Manchester nodes. JOIN-QUERY is executed using data sources located at Newcastle. The *goterm* and *protein_goterm* tables involved in this query are each exposed by three separate OGSA-DAI data services. This deployment replicates each table three times, and the OGSA-DQP coordinator chooses at random which data service to use to scan the tables when processing a query. The reason for replicating each of the tables is to reduce the likelihood of data sources becoming bottlenecks when multiple queries are processed concurrently. Experiments are performed with two different sets of available evaluators:

(1) A single evaluator is deployed at Manchester. This is equivalent to the wrapper-mediator approach.
(2) Nine evaluators are deployed on nodes at Newcastle, six of which are on the same nodes as the data sources. This configuration enables OGSA-DQP to parallelise the execution of joins over multiple evaluators.

For each of these deployment configurations, the average response time for queries is monitored as the frequency with which queries are submitted is increased (OGSA-DQP is able to process multiple queries concurrently using the same set of resources). The experiment is performed by periodically submitting JOIN-QUERY and calculating the average time taken to evaluate the queries over a fixed period. Initially, the experiment is performed with a client that waits 16 seconds between the submission of each query. Subsequently, the wait period is reduced by 2 seconds and the experiment is performed again. This process is repeated until the DQP infrastructure fails. For configuration 2, the optimiser will choose to parallelise each join using four evaluators. Two of these evaluators will reside on the nodes from which data is obtained in order to minimise the communication overhead. Metadata regarding the total memory and CPU speed of each node is made available to the DQP coordinator, however these properties are identical for the Newcastle nodes, meaning that the optimiser will randomly choose two of the evaluators used to execute each join from those available.

5.4.1 Results

Figure 11 plots the average response time for each query submission wait period for which the experiment was successfully performed. When the client waited less than 1 second between submitting each query, the evaluator used by the mediator-wrapper configuration ran out of memory and failed to process queries. The graph shows that when the query submission wait period

is 10 seconds or more, the response times of the mediator-wrapper approach (configuration 1) are less than those of the parallelised joins (configuration 2). For these query submission frequencies, there is no inter-query parallelism as the response time is less than the wait period. Under these conditions the mediator-wrapper architecture provides the better performance of the two configurations. Where the wait period is less then 10 seconds and the query evaluation infrastructure must process queries concurrently, the parallelised joins used in configuration 2 outperform the mediator-wrapper configuration. These results show that OGSA-DQP, with its support for partitioned parallelism and ability to schedule queries over collections of available execution nodes, can effectively reduce response times under these conditions and provide a scalable query evaluation infrastructure. In addition to the reduction of response times under such conditions, the parallelisation of the hash join, which caches its left input in main-memory, enhances scalability by providing a means of distributing memory usage over multiple nodes.

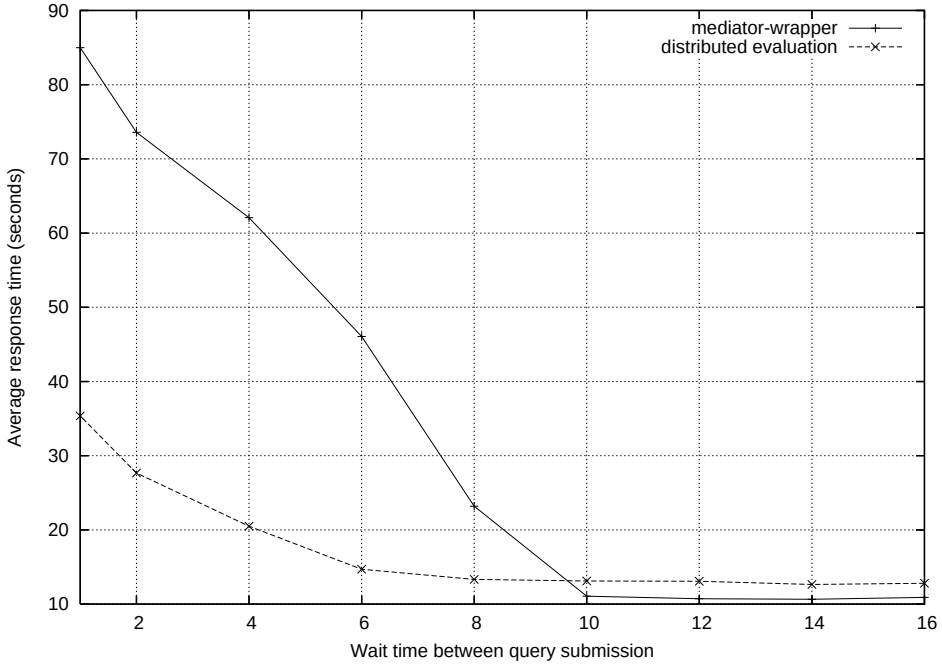

Fig. 11. Response times for analysis queries with different levels of parallelism

6 Related Work

Most early work on data Grids focused principally on the provision of infrastructures for managing and providing efficient access to file-based data [40]. The emphasis was therefore not on supporting structured data collections. However, the need to provide effective metadata catalogues for file archives gave rise to the use of database technology within data Grids (e.g., [16,40]),

and subsequently to the development of generic data access services, such as Spitfire [7] and OGSA-DAI [4]. This activity on metadata management, combined with the fact that an increasing number of the applications that use the Grid make extensive use of database technology, has increased awareness of the need to integrate database access interfaces with Grid middleware, both in terms of access to existing data resources and the wider management of data in Grid settings [2]. Community interest in Grid database services is reflected in the fact that the OGF has have provided interfaces for accessing databases in a Grid setting [5].

The distributed nature of Grid applications means that services to support coordinated use of Grid resources are important, and considerable attention has been given to functionalities for managing data derivation (e.g., [22]) and replication (e.g., [11]); such data Grid capabilities are also emerging in commercial offerings (e.g. [www.avaki.com]). However, such higher-level Grid data management functionalities are still targeted principally at file-based data, although the GRelC project has been developing an evolving collection of libraries and services to support database access and management in Grids [17].

The first proposal to use distributed query processing in a Grid setting was the Polar* proposal from the authors [49,50]. Polar* differs from the approach presented in this paper in that it is not service-based; in Polar*, Grid middleware is accessed using a Grid-enabled version of MPI [19]. The absence of the service-based context in Polar* means that connection to external databases and computational services is much less seamless than in the service-based setting.

Several projects use database language functionalities in pre-service-based Grids, where query languages express application requirements, which are then implemented by running jobs over Grid middlewares. For example, in GridDB [37] a functional language is proposed from which calls can be made to external programs or relational databases. These relational databases are typically used to represent internal state from a computation, and programs in the functional language are compiled for execution using Condor [36] to manage access to a cluster of computational resources. As such, GridDB can be seen as exploiting databases for information management within workflows expressed as functional programs, rather than as using queries as the way of expressing requests over multiple resources. A similarly named, but independent activity, is GridDB-Lite [43]. Like OGSA-DQP, in GridDB-Lite, data integration and analysis tasks are expressed as queries, although in a variation of SQL that makes explicit how data is to be partitioned over an analysis middleware. As such, like OGSA-DQP, the objective is to benefit from declarative requests over a Grid middleware. Architecturally the approach in GridDB-Lite seems rather different, in that the approach seems more bottom-up – identifying pat-

terns in the use of existing Grid libraries that can be captured in a declarative manner – rather than top-down – working out how an existing query language can be deployed for data and process integration in a service-oriented environment. In POQSEC [18], like GridDB and GridDB-Lite, queries expressed in a declarative language are compiled into scripts that run over an existing middleware. As such, an analogy can be drawn with Grid workflow engines such as Pegasus [15], in which abstract characterisations of application requirements are mapped onto lower level job and file descriptions for execution. This contrasts with OGSA-DQP, in which all resources are accessed by way of web services. All these proposals investigate the mapping of query functionalities onto Grid resources. In Grid query processing most such work has focused on extensions to classical distributed query processing architectures, although opportunities also exist for adapting other paradigms for use in a Grid setting. For example, HiSbase [47] uses application-specific mapping functions to allocate data to Grid nodes using a decentralized Peer-to-Peer model.

In a WS setting, structured data representations, at least in the form of XML Schemas, have been much more prominent from the start. In addition, vendors have been quick to integrate WS and data management products (e.g., [38,45]). One previous proposal for querying over collections of WSs is that of SkyQuery [39], which applies the classical wrapper-mediator architecture in a service-based setting. A variation of SkyQuery deploys WSs at each database store for handling metadata, performing queries, and cross matching partial results. However, the SkyQuery proposal is less ambitious than that presented here, in a number of respects: (i) the only services that contribute to query evaluation are the data sources – there is no query-specific allocation of evaluators, for example, to support evaluation of large joins or to reduce processing bottlenecks; (ii) the execution plan generated by the optimiser is a straightforward pipeline – there is no partitioned parallelism; and (iii) the query language supported is specialised for use with astronomical queries, and seems to assume that database nodes contain horizontal partitions of the overall database – there seem not to be generic facilities for joining data from multiple nodes, for example. Thus SkyQuery is an important early demonstration of the viability of WSs for supporting distributed query processing, but it lacks allocation of resources to match the needs of specific requests. This latter feature is central to the ethos of the Grid, in which computational resources are made shareable, and thus can be deployed flexibly to support changing user needs. An alternative approach for querying collections of WSs is provided by [52], which proposes a Web Service Management System (WSMS) that optimises pipelined execution plans over collections of Web services. Although this approach exploits the distribution of Web services over multiple nodes to provide parallelism, all other operations (such as joins) take place at the centralised WSMS. This approach eliminates the potential for the partitioned parallelism provided by OGSA-DQP.

How does the work presented here compare with other work on DQP, as surveyed in [35]? The principal differences derive from the context in which queries are executed. The aim of the current proposal is essentially the same as that of the developers of systems such as Garlic [33] and Kleisli [13], i.e., to support declarative query formulation over distributed data stores and analysis tools. However, the development of service-based Grids provides certain opportunities for the developers of DQP systems that were more elusive before. For example, WSs promise to make available comprehensive discovery and access facilities for distributed resources that ease their integration into federated architectures. We note that no custom-built wrappers were developed to support the bioinformatics application illustrated in this paper – generic OGSA-DAI data services were used to access the databases, and regular WSs were used as analysis components. OGSA-DAI provides not only access to the data in underlying sources, but also provides *ExtractDatabaseSchema* and *ExtractPhysicalSchema* activities, that provide details about sources that are used by OGSA-DQP in query compilation and optimization. This contrasts with both Garlic and Kleisli, where custom wrappers are constructed for interfacing the query engine to the external resources. In Garlic, the wrapper customization process allows different sources to provide different levels of service to the distributed query processor, whereas in OGSA-DQP we have chosen an off-the-shelf wrapper infrastructure with a view to minimising upfront configuration costs. In addition to the automatic extraction of source descriptions, we observe that the resources used to evaluate queries are allocated on a per-query basis, based on the anticipated needs of the request. Where requests require substantially greater resources to run efficiently, these can be allocated from those available on the Grid. This contrasts with both Garlic and Kleisli, where query evaluation is shared between the central query evaluator and the source wrappers, with no dynamic resource allocation.

There has been a significant amount of work on internet-scale query processing, with varying levels of similarity to OGSA-DQP. Perhaps the most similar is ObjectGlobe [8]. Although ObjectGlobe predates WSs, and thus service-based Grids, ObjectGlobe uses registries, dynamically allocated query engines and source wrappers in ways that have much in common with those in OGSA-DQP. The principal difference is that various functionalities were developed specifically to support ObjectGlobe that are provided in a generic way as part of a service-based Grid middleware. As such, OGSA-DQP can be seen as indicating how the requirements that motivated the development of ObjectGlobe can be supported within a service-based Grid. Various other proposals make assumptions that significantly affect the way a query processing technology is deployed or used. For example, in [44] and [31], query processing takes place over peer-to-peer networks, in which there is no global metadata, data sources arrive and depart organically during the evaluation of a query, and partial results are accepted as the norm. Approaches also exist for executing queries over data streams (e.g. [41,10]) where continuous queries may need to

be evaluated incrementally. OGSA-DQP and ObjectGlobe both deploy query components on potentially widely distributed computational nodes, but retain conventional query and data model semantics.

Several projects have used OGSA-DQP as a starting point for research into more experimental aspects of distributed query processing, or as a platform for data integration in scientific applications. In distributed query processing, extensions to the parallel query evaluator have been developed that: (i) support fault tolerance in the context of failures to evaluator nodes by maintaining intermediate query results in upstream caches until the data has been fully processed by downstream nodes [51]; (ii) adapt with a view to reducing load imbalance by dynamically changing the distribution of work across parallel query partitions [27]; and (iii) dynamically deploy evaluator web services as required to support query evaluation [42]. Furthermore, as data resources are accessed via OGSA-DAI, which in turn is designed to be extensible with repect to the sorts of data resource accessed (e.g. [46]), OGSA-DQP has been able to be extended to support different categories of data resource, including XML data resources and web-based data resources [53]. The focus on this paper has been narrower, with a view to detailing the stable features that form the public release of OGSA-DQP, and that have served as a starting point for these more preliminary investigations. In terms of applications, OGSA-DQP has been applied to support the intergration of proteome [55] and Epidemiology data resources [1].

7 Conclusions

WSs, in particular in conjunction with the resource access and management facilities of Grid computing, show considerable promise as an infrastructure over which distributed applications in e-business and e-science can be developed. However, to date, the emphasis has been on the development of core middleware functionalities, such as for service description, discovery and access. Extensions to support the coordinated use of such services, for example using distributed transactions [9] or workflow languages [12], are becoming more widely adopted. This paper seeks to contribute to the corpus of work on higher-level services by demonstrating how techniques from distributed query processing can be deployed in a service-based Grid. The proposal is service-based in two respects:

- Queries are written with respect to and evaluated over distributed resources discovered and accessed using WSs. This is important because it is as yet far from clear how best to orchestrate collections of services in data-intensive Grid applications. Although it is likely that workflow languages will have a prominent role, DQP offers system-supported optimisation of declarative re-

quests with implicit parallelism, a combination that should yield significant programmer productivity and performance benefits for large-scale, data intensive applications. As such, we believe that service-based architectures stand to benefit significantly from DQP. The proposal made in this paper is the most comprehensive to date for a distributed query processor that acts over services.

- The query processor has been designed and implemented as a collection of cooperating services, which is important because although service-based Grids have found widespread support within the academic and industrial Grid community, there are as yet few examples of higher-level services developed using them. This proposal can be seen to provide important validation of service-based Grids for developing higher-level functionalities. Furthermore, it has been shown how the combination of dynamic computational resource allocation can be used to match the requirements of a distributed query to the resources available in a heterogeneous distributed environment. In addition, experiments have illustrated the performance benefits in this context of both pipelined and partitioned parallelism in both local and wide area networks. As such, we believe that DQP stands to benefit significantly from the availability of service based Grids. The proposal made in this paper is much the most comprehensive to date for a distributed query processor that uses service-based Grids in its implementation.

Acknowledgements: This work has been supported by the UK e-Science programme, whose support we are pleased to acknowledge. Several coworkers have contributed to our understanding of query processing in service-based settings, including Nedim Alpdemir, Malcolm Atkinson, Desmond Fitzgerald, Anastasios Gounaris and Jim Smith.

References

[1] J. Ainsworth, R. Harper, I. Juma, and I. Buchan. Psygrid: Applying e-science to epidemiology. *19th IEEE Symposium on Computer-Based Medical Systems*, pages 727–732, 2006.

[2] G. Aloisio, M. Cafaro, and S. Fiore. The grid-dbms: Towards dynamic data management in grid environments. In *Proc. Intl. Conf. on Information Technology: Coding and Computation*, pages 199–204. IEEE Press, 2005.

[3] M.N. Alpdemir, A. Mukherjee, N.W. Paton, P. Watson, A.A.A. Fernandes, A. Gounaris, and J. Smith. Service-based distributed querying on the grid. In *Proc. 1st Int. Conf. Service-Oriented Computing*, pages 467–482. Springer, 2003.

[4] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N.P. Chue Hong, B. Collins, N. Hardman, A.C. Hulme, A. Knox, M. Jackson, A. Krause, S. Laws,

J. Magowan, N.W. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead. The design and implementation of Grid database services in OGSA-DAI. *Concurrency and Computation: Practice and Experience*, 17:357–376, 2005.

[5] M. Antonioletti, A. Krause, N. W. Paton, A. Eisenberg, S. Laws, S. Malaika, J. Melton, and D. Pearson. The ws-dai family of specifications for web service data access and integration. *SIGMOD Rec.*, 35(1):48–55, 2006.

[6] M. Atkinson, A.L. Chervenak, P. Kunszt, I. Narang, N.W. Paton, D. Pearson, A. Shoshani, and P. Watson. Data Access, Integration and Management. In *The Grid 2: Blueprint for a New Computing Infrashructure*, pages 391–429. Morgan-Kaufmann, 2004.

[7] W. H. Bell, D. Bosio, W. Hoschek, P. Kunszt, G. McCance, and M. Silander. Project Spitfire - Towards Grid Web Service Databases. In *Global Grid Forum 5*, 2002.

[8] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, and S. Seltzsam ans K. Stocker. ObjectGlobe: Ubiquitous Query Processing on the Internet. *VLDB Journal*, 10(1):48–71, 2001.

[9] F. Cabrera et al. Web Services Transaction (WS-Transaction). Technical report, IBM developerWorks Report, http://www-106.ibm.com/developerworks/library/ws-transpec/, 2002.

[10] M. Cherniack, H. Balakrishnan, and M. Balazinska. Scalable distributed stream processing. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, California, January 2003.

[11] A. Chervenak et al. Giggle: A Framework for Constructing Scaleable Replica Location Services. In *Proc. Supercomputing*. IEEE Press, 2002.

[12] F. Curbera et al. Business Process Execution Language for Web Services. Technical report, IBM developerWorks Report, http://www-106.ibm.com/developerworks/library/ws-bpel/, 2002.

[13] S. B. Davidson, J. Crabtree, B. P. Brunk, J. Schug, V. Tannen, G. C. Overton, and C. J. Stoeckert. K2/Kleisli and GUS: Experiments in Integrated Access to Genomic Data Sources. *IBM Systems Journal*, 40(2):512–531, 2001.

[14] S. de F. Mendes Sampaio, N. W. Paton, J. Smith, and P. Watson. Measuring and modelling the performance of a parallel odmg compliant object database server. *Concurrency and Computation: Practice and Experience*, 18(1):63–109, 2006.

[15] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[16] P. Dinda and B. Plale. A Unified Relational Approach to Grid Information Services. Technical Report GWD-GIS-012-1, Global Grid Forum, 2001.

[17] S. Fiore, A. Negro, S. Vadacca, M. Cafaro, M. Mirto, and G. Aloisio. Advanced Grid DataBase Management with the GRelC Data Access Service . In *Parallel and Distributed Processing and Applications*, pages 683–694. Springer, 2007.

[18] R. Fomkin and T. Risch. Framework for querying distributed objects managed by a grid infrastructure. In *Data Management in Grids (DMG)*, pages 58–70. Springer, 2005.

[19] I. Foster and N. Karonis. A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems. In *Proc. Supercomputing*. IEEE Press, 1998.

[20] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *IEEE Computer*, 35(6):37–46, 2002.

[21] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. J. Supercomputer Applications*, 15(3), 2001.

[22] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration. In *Proc. CIDR*, 2003.

[23] H. Garcia-Milina, J.D. Ullman, and J. Widom. *Database System Implementation*. Morgan Kaufmann, 2000.

[24] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to Web Services Architecture. *IBM Sys. Journal*, 41(2):170–177, 2002.

[25] A. Gounaris, C. Comito, R. Sakellariou, and D. Talia. A service-oriented system to support data integration on data grids. In *CCGRID*, pages 627–635, 2007.

[26] A. Gounaris, R. Sakellariou, N. W. Paton, and A. A. Fernandes. A novel approach to resource scheduling for parallel query processing on computational grids. *Distrib. Parallel Databases*, 19(2-3):87–106, 2006.

[27] A. Gounaris, J. Smith, N. W. Paton, R. Sakellariou, A. A. A. Fernandes, and P. Watson. Adapting to changing resource performance in grid query processing. In *Data Management in Grids (DMG)*, pages 30–44. Springer, 2005.

[28] G. Graefe. Encapsulation of Parallelism in the Volcano Query Processing System. In *Proc. SIGMOD*, pages 102–111, 1990.

[29] S. Graham and J. Treadwell (editors). Web service resource properties 1.2. Technical report, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf, 2006.

[30] M.A. Harris et al. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Research*, 32(1):D258–261, 2004.

[31] R. Huebsch, J.M. Hellerstein, N. Lanham, B. Thau Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *VLDB*, pages 321–332, 2003.

[32] M. Jackson et al. Performance Analysis of the OGSA-DAI Software. In *Proc. e-Science All Hands Conference*, 2004.

[33] V. Josifovski, P. Schwarz, L. Haas, and E. Lin. Garlic: A New Flavor of Federated Query Processing for DB2. In *Proc. SIGMOD*, pages 524–532, 2002.

[34] W. Kim, I. Choi, S. K. Gala, and M. Scheevel. On resolving schematic heterogeneity in multidatabase systems. *Distributed and Parallel Databases*, 1(3):251–279, 1993.

[35] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[36] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor - A Hunter of Idle Workstations. In *Proc. 8th International Conference on Distributed Computing Systems*, pages 104–111. IEEE Press, 1988.

[37] D.T. Liu and M.J. Franklin. GridDB: A Data-Centric Overlay for Scientific Grids. In *Proc. VLDB*, pages 600–611. Morgan-Kaufmann, 2004.

[38] S. Malaika, C.J. Nelin, R. Qu, B. Reinwald, and D. C. Wolfson. DB2 and Web Services. *IBM Systems Journal*, 41(4):666–685, 2002.

[39] T. Malik, A.S. Szalay, T. Budavari, and A.R. Thakar. SkyQuery: A Web Service Approach to Federate Databases. In *Proc. CIDR*, 2003.

[40] R. W. Moore, C. Baru, R. Marciano, A. Rajasekar, and M. Wan. Data-Intensive Computing. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastrcuture*, chapter 5, pages 105–129. Morgan Kaufmann, 1999.

[41] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 245–256, Asilomar, California, January 2003.

[42] A. Mukherjee and P. Watson. Adding dynamism to ogsa-dqp: Incorporating the dynasoar framework in distributed query processing. In *Euro-Par Workshops*, pages 22–33, 2006.

[43] S. Narayanan, T.M. Kurc, and J. Saltz. Database Support for Data-Driven Scientific Applications in the Grid. *Parallel Processing Letters*, 13(2):245–271, 2003.

[44] V. Papadimos, D. Maier, and K. Tufte. Distributed query processing and catalogs for peer-to-peer systems. In *Proceedings of CIDR 2003, First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA*, 2003.

[45] R. M. Riordan, editor. *Microsoft ADO.NET Step by Step.* Microsoft Press, 2002.

[46] A. Sánchez, M. S. Pérez, K. Karasavvas, P. Herrero, and A. Pérez. Mapfs-dai, an extension of ogsa-dai based on a parallel file system. *Future Generation Comp. Syst.*, 23(1):138–145, 2007.

[47] T. Scholl, B. Bauer, B. Gufler, R. Kuntschke, A. Reiser, and A. Kemper. Scalable community-driven data sharing in e-science grids. *Future Generation Comp. Syst.*, 2008. doi:10.1016/j.future.2008.05.006.

[48] B. Seshasayee, K. Schwan, and P. Widener. Soap-binq: High-Performance SOAP with continuous quality management. In *Proc. ICDCS*, pages 158–165, 2004.

[49] J. Smith, A. Gounaris, P. Watson, N. W. Paton, A. A. A. Fernandes, and R. Sakellariou. Distributed Query Processing on the Grid. In *Proc. Grid Computing 2002*, pages 279–290. Springer, LNCS 2536, 2002.

[50] J. Smith, A. Gounaris, P. Watson, N.W. Paton, A.A.A. Fernandes, and R. Sakellariou. Distributed query processing on the grid. *Intl. J. High Performance Computing Applications*, 17(4):353–368, 2003.

[51] J. Smith and P. Watson. Fault-tolerance in distributed query processing. In *IDEAS*, pages 329–338. IEEE Press, 2005.

[52] U. Srivastava, K. Munagala, J. Widom, and R. Motwani. Query optimization over web services. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pages 355–366. VLDB Endowment, 2006.

[53] Said Mirza Pahlevi Steven Lynden and Isao Kojima. Service-based data integration using ogsa-dqp and ogsa-webdb. In *Proc. 9th IEEE/ACM International Conference on Grid Computing (Grid 2008)*. IEEE Press, 2008.

[54] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643–662, 2001.

[55] L. Zamboulis, H. Fan, K. Belhajjame, J. A. Siepen, A. C. Jones, N. J. Martin, A. Poulovassilis, S. J. Hubbard, S. M. Embury, and N. W. Paton. Data access and integration in the ispider proteomics grid. In *Data Integration in the Life Sciences (DILS)*, pages 3–18. Springer, 2006.

[56] L. Zamboulis, N. Martin, and A. Poulovassilis. Query Processing and Optimization in Integrated Heterogeneous Grid Resources. Technical Report BBKCS-08-05, Birkbeck College, University of London, 2008.