

OGSA-DQP: A Service-Based Distributed Query Processor for the Grid

M. Nedim Alpdemir¹, Arijit Mukherjee², Anastasios Gounaris¹,
Norman W.Paton¹, Paul Watson², Alvaro A.A. Fernandes¹, Jim Smith²

(1) Department of Computer Science
University of Manchester
Oxford Road, Manchester M13 9PL
United Kingdom
{alpdemim|norm|alvaro|gounaris}@cs.man.ac.uk

(2) School of Computing Science
University of Newcastle upon Tyne
Newcastle upon Tyne NE1 7RU
United Kingdom
{Arijit.Mukherjee|Paul.Watson|Jim.Smith}@ncl.ac.uk

Abstract

The Grid is an emerging infrastructure that supports the discovery, access and use of distributed computational resources. The emergence of a service-oriented view of hardware and software resources on the grid raises the question as to how database management systems and technologies can best be deployed or adapted for use in such an environment.

We argue that distributed query processing (DQP) can provide effective declarative support for service orchestration, and we describe an approach to service-based DQP (OGSA-DQP) on the Grid that supports queries over Grid Data Services (GDS) provided by OGSA-DAI project, and over other services available on the Grid, thereby combining data access with analysis; uses the facilities of the OGSA to dynamically obtain the resources necessary for efficient evaluation of a distributed query; adapts techniques from parallel databases to provide implicit parallelism for complex data-intensive requests; and uses the emerging standard for GDSs to provide consistent access to database metadata and to interact with databases on the Grid.

The service-based Distributed Query Processor is itself cast as a service referred to here as Grid Distributed Query Service (GDQS). In addition, OGSA-DQP employs another service for query evaluation referred to here as Grid Query Evaluation Service (GQES). As such, OGSA-DQP implements a service orchestration framework in two sense: both in terms of the way its internal architecture handles the construction and execution of distributed query plans and in terms of being able to query over data and analysis resources made available as services.

1 Introduction

Service-based approaches (such as Web Services and the Open Grid Services Architecture) have gained considerable attention recently for supporting distributed application development in e-business and e-science. The service-based approach seems to many a good solution to the problem of modelling a virtual organisation as a distributed system.

The Open Grid Services Architecture (OGSA) [3, 8], build upon and extend the service-oriented architecture and technologies first proposed for Web Services (WSs) [4]. The OGSA proposes conventions and interfaces for a *Grid Service*, a (potentially transient) stateful service instance supporting reliable and secure invocation, lifetime management and notification. This allows the dynamic creation of service instances on computational resources that are discovered and allocated as, and when, they are needed.

Although the initial emphasis in Grid computing was on file-based data storage [6], the importance of structured data management to typical Grid applications is becoming widely recognised, and several proposals have been made for the development of Grid-enabled database services (e.g. Spitfire [1], OGSA-DAI [www.ogsa-dai.org.uk]). Ongoing work in the Database Access and Integration Services Working Group of the Global Grid Forum is developing a proposal for a standard service-based interface to relational and XML databases in the OGSA setting [www.gridforum.org/6_DATA/dais.htm]. This specification uses the XML-based Web Services Description Language (WSDL) to specify the interface that should be supported by a *Grid Database Service* (GDS).

The system presented here, namely OGSA-DQP,

is a proof of concept implementation of a service-based distributed query processor on the grid, that aims to exploit the service-oriented middleware provided by OGSA-DAI and OGSA reference implementation, Globus Toolkit 3 (GT3), by plugging into the port types defined by the constituent services of those frameworks.

OGSA-DQP supports the evaluation of queries expressed in a declarative language over one or more existing services. These services are likely to include mainly database services, but may also include other computational services. As such, OGSA-DQP supports service orchestration and can be seen as complementary to other infrastructures for service orchestration, such as workflow languages. In principle, OGSA-DQP can be used in any Grid application that aims to integrate and analyse structured data collections. OGSA-DQP uses the emerging standard for GDSs to provide consistent access to database metadata and to interact with databases on the Grid. Notably, it also adapts techniques from parallel databases to provide implicit parallelism for complex data-intensive requests.

This paper aims to provide a user-oriented view of OGSA-DQP, rather than delving into the details of its design and implementation. Therefore, after the following section which gives an overview of the OGSA-DQP system, Section ?? provides a step-by-step walk-through that illustrates the usage of the system, via a GUI client.

2 Overview

OGSA-DQP is an example of a high level data integration framework. It is also an example of a service orchestration framework in that it coordinates the incorporation of external analysis services into data retrieval. As shown in Figure 1, OGSA-DQP

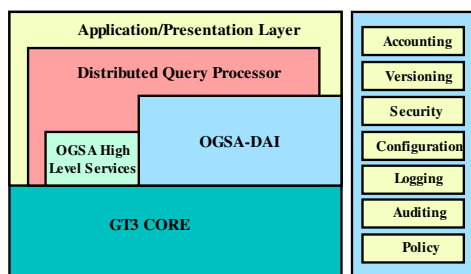


Figure 1: Layered Architecture of OGSA-DQP

has a layered architecture. It uses services provided by the OGSA-DAI framework to access potentially heterogeneous data sources. The OGSA-DAI frame-

work provides *Grid Data Services (GDS)* to give access to (potentially heterogenous) stored collections of structured data managed by database management systems implementing standard data models. At a lower level, lies GT3, the OGSA reference implementation which is used both by OGSA-DQP and OGSA-DAI for service instance creation, service state access, and lifetime management of the service instances.

OGSA-DQP provides two services to fulfil its functions: The Grid Distributed Query Service (GDQS) and the Grid Query Evaluation Service (GQES). The implementation of the GDQS builds on our previous work on the Polar* distributed query processor for the Grid [7] by encapsulating its compilation and optimisation functionality. The GDQS provides the primary interaction interfaces for the user and acts as a coordinator between the underlying query compiler/optimiser engine and the GQES instances. The GQES, on the other hand, is used to evaluate (i.e. execute) a query sub-plan assigned to it by the GDQS. The number of GQES instances and their location on the grid is specified by the GDQS based on the decisions made by the query optimiser and represented as an execution schedule for query partitions (i.e. sub-plans). GQES instances are created and scheduled dynamically, and their interaction is coordinated by the GDQS.

It is important to note that GDQS relies on OGSA-DAI port types and extends those port types with additional functionality (i.e. with an additional port type). Consequently, much of the interaction patterns supported by a standard GDS are also supported by a GDQS. This brings about uniformity in the users' perception of the GDQS as a data integration service.

The interaction of a client with the GDQS can roughly be divided into two phases:

1. Set-up phase
2. Query evaluation and result delivery phase

The following sections provide an overview of the operations that take place during each phase.

2.1 GDQS Set up

The GDQS implements an additional port type to enable the users to specify a set of data sources and analysis services to be used for queries. The Grid Distributed Query (GDQ) port type defines a single operation, namely `importSchema`, for that purpose. The `importSchema` operation is used to prepare the GDQS for query submission. Preparing the GDQS for query submission involves identifying the data sources and the analysis services to be used in the

query. The data sources have to be identified by the Grid Data Service Factory (GDSF) handles that wrap those data sources. The analysis services have to be identified by URLs which point to the WSDL documents describing those services. Both the GDSF handles and the WSDL URLs have to be included in a list, inside an XML document, which should then be supplied as the only parameter to the `importSchema` operation.

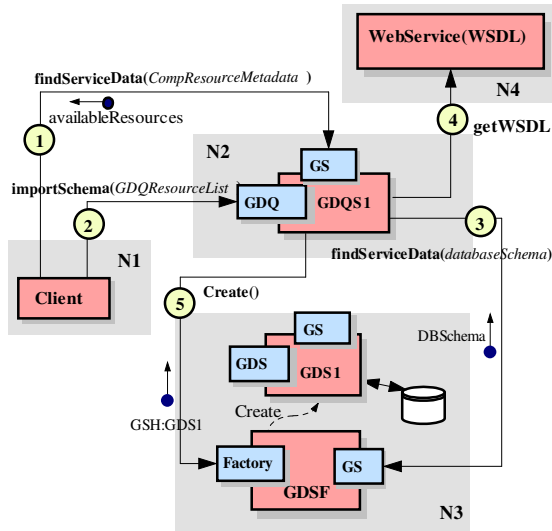


Figure 2: Interactions during the GDQS Set-up Phase

Figure 2 illustrates interactions that take place during a typical setup phase. The client first obtains a list of available resources from the GDQS instance by querying its Service Data Element (SDE) whose name is `CompResourceMetadata` (interaction 1). This call uses the Grid Service (GS) port type `findServiceData` operation to access the service state. Then the client calls the `importSchema` operation on the GDQ port type and provides a list of resources (interaction 2). The GDQS then interacts with the specified GDS Factories to obtain the schemas of the databases they wrap, and with service WSDL URLs to obtain the WSDL documents of the analysis services (interactions 3 and 4). The GDQS also creates GDS instances so that the query evaluators can access data during query execution (interaction 5).

Note that schema conflict resolution is not supported during schema import; the imported schemas are simply accumulated and maintained locally.

2.2 Query Submission

The GDQS accepts query submissions in the form of OQL [2] queries via the Grid Data Service (GDS) port type defined by OGSA-DAI. The query is embedded within an XML document called a *GDS perform document* [5]. The perform document can be configured such that the results are delivered synchronously to the client, or streamed and pulled asynchronously by the client or pushed to another service. These alternative interaction modes can be specified by a set of activities embedded in the perform document. For a detailed explanation of how these can be achieved see [5]. In this paper only the synchronous mode of delivery is described, even though asynchronous delivery is also exploited in the implementation.

Figure 3 illustrates the interactions that take place when a query is received and processed by a GDQS instance. The client submits a query via `GDS perform` operation (interaction 1). The query contained in the perform document is compiled and optimised into a distributed query execution plan, whose partitions are scheduled for execution at different GQESs (this is not explicitly shown in the figure). The GDQS then uses this information to create GQES instances on their designated execution nodes (interaction 2). Next, the GDQS hands over to each GQES the sub-plan assigned to it (interaction 3). This initiates the query execution process whereby some of the GQES instances interact with other GDS instances to obtain data (interaction 4). Eventually, the results start to propagate across GQES instances and, ultimately, reach the client (interaction 5).

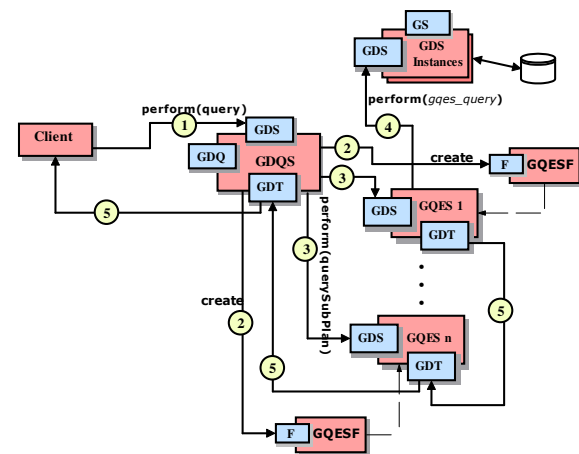


Figure 3: An Overview of Interactions During Query Execution

The next section illustrates how OGSA-DQP can be used. For this purpose, an example query submission

sion procedure is walked through in detail.

3 Using OGSA-DQP

To illustrate the GDQS set-up and query submission phases introduced above we use the setting illustrated in Figure 4. The experimental setting involves five separate machines, four of which are located on the same campus and one in a campus in another city. Individual machines are identified by node numbers and denoted by a shaded square. Note that three of the nodes have GDS Factories installed on them to wrap the three databases located on those nodes. The databases are hosted by MySQL DBMS. The table definitions comprising the schema of each database are given below the shaded boxes.

An analysis service, namely `EntropyAnalyzerService` is also deployed on one of the nodes, and will be used to illustrate how OGSA-DQP can combine data analysis with data retrieval.

The GDQS Factory is installed on a separate node, although it is conceivable that it might have existed on one of the other nodes alongside GDS Factories. Note, also, that Grid Query Evaluator Factories are installed on all of the nodes to enable the exploitation of every machine available for the execution of the query.

The following subsections walk the reader through the steps required to submit and execute queries in the setting explained above.

3.1 Service Discovery and Instance Creation

3.1.1 Locating the Factory

As implied by the service oriented architectures, the initial step to any client-service interaction is the discovery of desired services in a public registry. Note, however, that here the discovered entity is not the service itself but a factory that can be used to create the actual service instance. OGSA-DAI defines the notion of a Grid Data Service Registry (GDSR), which is based on the OGSA Service Group Registry concept [8]. A GDSF registers itself with at least one, but potentially multiple, registries when it is initialised. Those registries are specified as part of the GDSF configuration process. The GDQSF, obeys the same pattern and registers itself to the GDSRs specified in its configuration. Figure 5 illustrates this. Typically the registry would range over a Virtual Organisation (VO) known to the user, and would be identified by a Grid Service Handle (GSH). The GUI

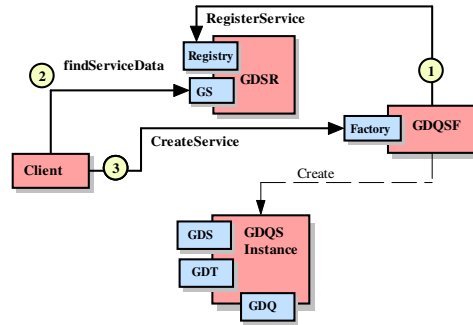


Figure 5: Discovering the GDQS Factory and Instance Creation

Client implemented for OGSA-DQP provides a dialog where it is possible to enter the GSH of a registry and inspect its content (i.e., obtain the list of registered services). Figure 6 illustrates the screen shot

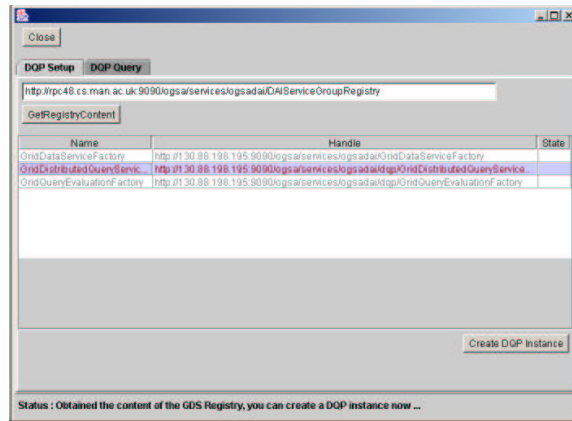


Figure 6: Discovering the GDQS Factory and Instance Creation

of the dialog. Note that the handle of the GDQSF is highlighted to indicate that GDQS Factory was located.

3.1.2 Creating a GDQS Instance

As the OGSA adopts a model where interactions with services follow a stateful instance approach, once the factory is located, the next step is to create an instance. Using the OGSA-DQP GUI Client this can be done by pressing the appropriate button, shown in Figure 6. When a GDQS instance is created, several initialisation steps take place:

- The GDQSF passes configuration information to the instance it creates. Some information, such as the credentials of the user, OGSA-DAI activity types supported by the instance,

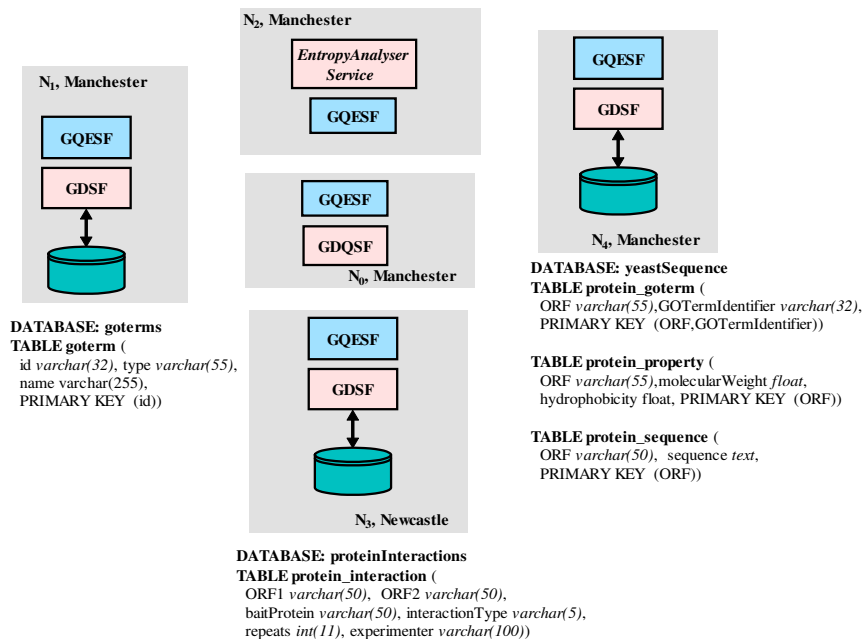


Figure 4: An Example distributed Setting used for testing OGSA-DQP

is inherited from the standard GDSF behaviour. Other information, such as the set of XSL transformations required for transforming externally obtained documents into a form understandable by the query optimiser and the location of the query optimiser engine is specific to GDQS.

- An XML document that contains information about the computational characteristics of the nodes being made available for processing is loaded by the GDQS instance. This information will be obtained dynamically in future versions, but currently it is statically loaded.
- Some of the Service Data Elements (SDEs) are initialised with the data collected during the initialisation.

Once the instance is created it is then ready for setting it up to acquire resources required for the queries, which is explained in the next section.

It is worth noting that GDQS instances are created per-client and can handle multiple queries but are still relatively short-lived entities. This model is also applicable to the lifetime of the GDS instances created and managed by the GDQS.

Clearly, other approaches to lifetime management are possible, each having particular advantages and disadvantages. For instance, the GDQS could be implemented to serve multiple queries from multiple users for a long period of time. In that case the cost

of setting up the service would be reduced, at the expense of somewhat increased complexity due to the need to manage multiple coexistent query requests and the resources allocated for their executions. On the other hand, the GDQS instance could be designed to be a per-query, short-lived entity, in which case the cost of setting up the service might come to constitute a considerable proportion of the total cost of the service provided by the instance. The approach adopted for our particular implementation (i.e., an instance per-user that is capable of responding to multiple requests by that user), avoids the complexity of multi-user interactions while ensuring that the set-up cost is unlikely to be the dominating one.

3.2 Preparing the GDQS for Query Execution

As pointed out in Section 2.1, preparing a GDQS for query submission involves identifying the data sources and the analysis services to be used in the query. The GDQS instance helps the user in identifying the available resources by exposing the computational resource metadata loaded during its initialisation as an SDE. The client can query the `ComResourceMetadata` SDE to see the available resources. The user can then compile a list of the data resources (identified by GDSF handles) and the analysis resources (identified by WSDL URLs) required for a particular query session, and hand them to the

GDQS using the `importSchema` operation. For the example setting described in Section 3 and illustrated in Figure 4, the input to the `importSchema` operation would be the following XML document:

```
<GDQDataSourceList>
  <importedDataSource>
    <GDSFactoryHandle>
      http://machine1.cs.man.ac.uk:8080
      /ogsa/services/ogsadai
      /GridDataServiceFactory
    </GDSFactoryHandle>
    <GDSFactoryHandle>
      http://machine2.cs.man.ac.uk:8080
      /ogsa/services/ogsadai
      /GridDataServiceFactory
    </GDSFactoryHandle>
    <GDSFactoryHandle>
      http://machine1.ncl.ac.uk:8080
      /ogsa/services/ogsadai
      /GridDataServiceFactory
    </GDSFactoryHandle>
  </importedDataSource>
  <importedService>
    <wsdlURL>
      http://machine3.cs.man.ac.uk:9090/
      axis/services
      /EntropyAnalyserService?WSDL
    </wsdlURL>
  </importedService>
</GDQDataSourceList>
```

Once the list is submitted, the GDQS instance contacts the GDSFs provided in the list to extract the database schema of the data sources by querying the `databaseSchema` SDE exposed by those GDSFs. It also processes the WSDL documents pointed out by the provided WSDL URLs. Figure 7 is a snapshot of the GUI client screen with data sources and analysis service selected and ready for schema import. Initially only the top list is displayed and user is expected to select items from the list to add to the selected data source list. Similarly, the user can enter WSDL URLs to the provided edit box and add them to the selected services list.

Note that it is conceivable to take different approaches to GDQS setup, in regard with the extent of required interaction. The following are possible:

1. **Generic GDQS.** As described above a GDQS can start as an empty box, in which case the user needs to provide the full list of required resources for a particular query session.
2. **Partially configured GDQS.** It might be desirable to configure the GDQS factory so that the instance, when created, starts with an initial set of resources already acquired to enable the users to start querying immediately. In the case that the query session requires the addition

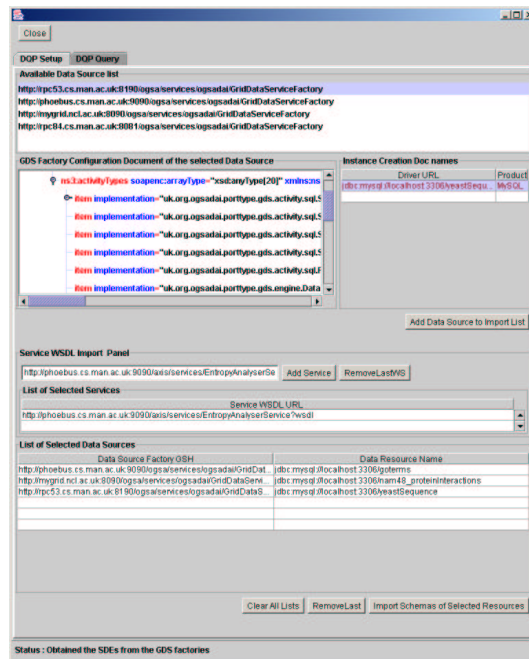


Figure 7: The GDQS Setup Dialog

of more resources, the user can add those resources using the `importSchema` operation as described above.

3. **Fully Configured GDQS.** For well-defined query requirements it might make sense to have fully configured GDQS factories (i.e., a special case of item 2), for frequent use within an organisation. Such GDQS Factories can be advertised in the VO registry with different names and additional metadata to indicate their capabilities in terms of the resources they can integrate.

Although only item 1 is supported in the current OGSA-DQP implementation, it is fairly easy to extend GDQS to support items 2 and 3 since the factory configuration schemes in OGSA-DAI are capable of incorporating such extensions in a well-defined way.

3.3 Submitting Query Requests

Before writing the query the user may wish to have access to the schemas of the data sources. To accommodate this need, GDQS exposes the metadata about the imported resources as an SDE, so the user can examine the database schema of the imported databases by querying the `importedSchemas` SDE.

As pointed out in Section 2.2, submitting the query requires the user to embed the OQL query into

a query request document, defined by the GDS specification. In the case of the OGSA-DQP GUI Client the query is simply entered as an OQL text and the request document is constructed on the fly. The following query is used as an example to illustrate the query submission procedure.

```
print select p.ORF, go.id,
           calculateEntropy(p.sequence)
from p in protein_sequences,
     go in goterms,
     pg in protein_goterms
where
  go.id=pg.GOTermIdentifier and
  p.ORF=pg.ORF and
  p.ORF like "YBL06%" and
  go.id like "GO:0000%";
```

The query contains two separate join operations each joining two extents (or tables in relational databases terms) from different databases (on different servers) and applies entropy analysis on sequences obtained from one of the extents using a web service. The `calculateEntropy` method is an operation defined by the `EntropyAnalyzerService`. Note that the parameter to this method is a column from `protein_sequences` extent. The query is embedded in the following GDS perform document:

```
<gridDataServicePerform>
  <documentation>
    This request submits an OQL query to
    GDQS to retrieve data from distributed
    data sources.
  </documentation>

  <oqlQueryStatement name="statement">
    <expression>
      print select p.ORF, go.id,
                 calculateEntropy(p.sequence)
      from p in protein_sequences,
           go in goterms,
           pg in protein_goterms
      where
        go.id=pg.GOTermIdentifier and
        p.ORF=pg.ORF and
        p.ORF like "YBL06%" and
        go.id like "GO:0000%";
    </expression>
    <webRowSetStream name="stOutput"/>
  </oqlQueryStatement>
</gridDataServicePerform>
```

Figure 8 shows the query execution pane of the GUI Client, with the results displayed in the lower pane and schema information displayed as an XML document in the top pane.

Table 1 gives an indication of the response times for various operations performed during the whole exercise. The top two rows in the table show the size



Figure 8: The GUI Client Query Submission Dialog

Table 1: Response Times for Different Phases

extent names	protein_goterms	goterms	protein_sequences
extent sizes (in rows)	16803	11369	6303
execution times			
GDQS Setup	Query Optimisation	Query Post Processing	Query Execution
1.3 sec	107 milisc	715 milisc	13 sec

of the three extents involved in the query in terms of the number of rows they contain. The bottom row provides the elapsed time during the main operations such as the service set-up (i.e. importing resource schemas), the query compilation and optimisation, query post processing and query execution. Note that the figures in the table are meant to provide a rough idea about the response times and should not be interpreted as the result of a thorough performance analysis.

4 Conclusion

There has not been much on service orchestration on the Grid; the system described here provides one approach to declarative support for service orchestration via dynamic resource discovery and allocation. OGSA-DQP provides an example of a high level

Grid Service Framework that is capable of combining data analysis with data integration. In fact, to our knowledge, it is the first service-based distributed query processor that demonstrates how a DQP can combine data access with data analysis.

From a user point of view, OGSA-DQP exemplifies the patterns of interaction with dynamically created, stateful services, which involves the discovery of service factories using registries, explicit creation of service instances via factories, and accessing the instance state via Service Data Elements.

As pointed out in Section 3.1.2, the instance lifetime model adopted for GDQS and GDSs aims to balance the cost of system set up with implied complexity in the system. Furthermore, the cost of GDQS set up can be compensated for by moving towards using partially or fully configured GDQS factories.

Although Section 3.3 provides response times of various operations during the GDQS setup and query execution, a detailed performance analysis needs to be done to closely examine the behaviour of the system and identify the bottlenecks in the infrastructure. This is a prioritized item in our future work list. It is, however, reasonable to expect that as replicated data sources and analysis services become readily available, the relative response time is likely to be even better since OGSA-DQP design is geared towards exploiting available grid resources to implement parallel execution.

Acknowledgements — The work reported in this paper has been supported by the UK e-Science Core Programme through the OGSA-DAI project and the myGrid project, and by the UK EPSRC grant GR/R51797/01. We are grateful for that support.

References

- [1] 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.
- [2] R. G. G. Cattell and D. K. Barry. *The Object Database Standard: ODMG 3.0*. Morgan Kaufmann, 2000.
- [3] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *IEEE Computer*, 35(6):37–46, 2002.
- [4] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to Web Services Architecture. *IBM Sys. Journal*, 41(2):170–177, 2002.
- [5] A. Krause, T. Sugden, and A. Borley. Grid Data Service. Technical report, OGSA-DAI, 2003. Document Identifier: OGSA-DAI-USER-UG-GDS-v4.1, July, 2003.
- [6] 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 Infrastructure*, chapter 5, pages 105–129. Morgan Kaufmann, 1999.
- [7] 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.
- [8] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. Open Grid Services Infrastructure (OGSI), Version 1.0. Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, 2003. June 27, 2003.