Adding Dynamism to OGSA-DQP: Incorporating the DynaSOAr Framework in Distributed Query Processing

Arijit Mukherjee and Paul Watson

School of Computing Science, Newcastle University,
Claremont Tower, Claremont Road, Newcastle Upon Tyne, United Kingdom
{Arijit.Mukherjee,Paul.Watson}@ncl.ac.uk
http://www.cs.ncl.ac.uk

Abstract. OGSA-DQP is a Distributed Query Processing system for the Grid. It uses the OGSA-DAI framework for querying individual databases and adds on top of it an infrastructure to perform distributed querying on these databases. OGSA-DQP also enables the invocation of analysis services, such as **Blast**, within the query itself, thereby creating a form of declarative workflow system. DynaSOAr is an infrastructure for dynamically deploying web services over a Grid or a set of networked resources. The DynaSOAr view of grid computing revolves around the concept of services, rather than jobs where services are deployed on demand to meet the changing performance requirements. This paper describes the merging of these two frameworks to enable a certain amount of dynamic deployment to take place within distributed query processing.

Keywords: Dynamic deployment, Web Service, Grid, distributed query processing.

1 Introduction

OGSA-DQP[1], [2] is a publicly available service-oriented distributed query processor for the Grid. It provides distributed query functionality on databases spread over the Grid using the commonly used service for data access and integration, OGSA-DAI[3]. OGSA-DQP supports the evaluation of queries expressed in a declarative fashion over one or more services, including data access services and external analysis services. It can be seen as complimentary to other service orchestration mechanisms, such as workflow languages.

Because the services can be potentially located on computational resources distributed across the internet, communication costs can play a major role in the performance of the system. Co-locating the query evaluation service and the analysis service with the data, even with an *on-the-fly* deployment may prove to be potentially beneficial in the long run, especially when frequent, long-running queries are executed. DynaSOAr[5] is a framework, which enables such dynamic deployment of services on available computational nodes. An enhanced version of OGSA-DQP has been created which incorporates DynaSOAr concepts. This paper briefly describes the DynaSOAr architecture, and its use within the OGSA-DQP context. Experimental results and analyses show how DynaSOAr may benefit service-oriented distributed query processing by moving the analysis and data retrieval code near the actual data.

The paper is organized as follows: Section 2 provides a brief introduction to the OGSA-DQP concept and functionality. Section 3 describes the DynaSOAr architecture in brief, followed by the use of DynaSOAr infrastructure within the OGSA-DQP context in Section 4. The experimental setup, results and analysis are covered in Section 5. Related works are discussed in Section 6, current and future directions in Section 7, with conclusions in Section 8.

2 Brief Description of OGSA-DQP

OGSA-DQP is composed of two major services (i) Grid Distributed Query Service (GDQS) and (ii) Query Evaluation Service (QES). The GDQS is implemented as an extension to the standard OGSA-DAI service, and is deployed as an OGSA-DAI data service with an exposed data service resource[1]. The DQP data service resource thus exposed, supports querying over a set of OGSA-DAI data services, each wrapping a database on some computational node. It also supports the invocation of analysis services over the query results. An example of a typical query, in OQL, supported by OGSA-DQP is as follows:

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

In this example, the query spans over three databases (protein_sequence, goterm and protein_goterm) which can be distributed over a large geographical area, and an analysis service exposed as a Web Service is also invoked on each sequence element. Based on the schema and WSDL imported from the data and the analysis services and the resources available to it, a query compiler/optimizer component, Polar*[6], generates a parallel query plan, which is partitioned into sub-plans. These sub-plans are distributed to the participating evaluation services each of which is responsible for evaluating the sub-plan assigned to it and conveying the result back either to the root partition or other evaluation services. Finally, the result is collected at the node evaluating the root partition and sent to the GDQS and hence to the consumer.

[1] A data service resource implements the core OGSA-DAI functionality. It accepts perform documents from data services, parses and validates them, executes the data-related activities specified within them and constructs response documents. It can also cache data for retrieval by third-parties (if the data service resource is configured to support asynchronous data delivery). Data service resources are accessed via data services.[4]

3 DynaSOAr Architecture

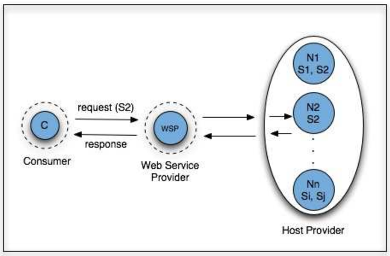
DynaSOAr is a framework, which provides a generic infrastructure for deploying web services as and when required, on available nodes. DynaSOAr achieves this dynamic deployment by processing the incoming consumer request at different levels between two different components, namely a *DynaSOAr Web Service Provider* and a *Host Provider*, with a defined interface between them.

– The *DynaSOAr Web Service Provider* is the entity with which consumers interact. It advertises the services it can provide, receives SOAP messages from consumers requesting a service from a particular endpoint associated with the message, and is responsible for arranging the processing of the request. The *Web Service Provider* achieves this by choosing an appropriate *Host Provider* and forwarding the message to it with any associated Quality of Service (QoS) parameters and an added element in the message header identifying a *software repository* where the service code can be found in case a dynamic deployment is required.
– The *Host Provider* is responsible for controlling the computational resources, such as a cluster or a grid, on which services can be deployed, and requests for those services can be processed. It accepts the SOAP messages forwarded by the *Web Service Provider* on behalf of the services hosted by it, and sends back any response generated after processing the request.

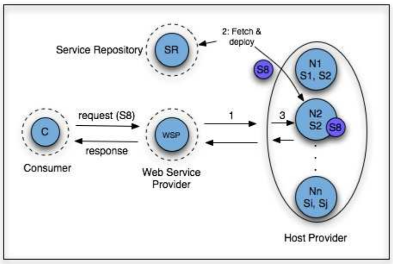
When a message reaches the *Host Provider*, there can be two different interaction patterns depending on whether or not the requested service is already deployed on the node -

1. If the service is already deployed on the computational node where the request is to be processed, then the *Host Provider* routes the SOAP message to the service on that node. In Figure 1a, the consumer makes a request for service S2, which is already deployed on node *N1* and *N2*. Based on the current information about the system load, the *Host Provider* routes the request to the lightly loaded node *N2* where the request is processed and the response is sent back.
2. The second case is where the consumer makes a request for a service, which is not already deployed on any of the available nodes, such as the request for service S8 sent by the consumer in Figure 1b. In this case, a decision is made about the target node where the service is to be deployed and the message is forwarded to that node. The node downloads the service code from the *software repository*, deploys the service dynamically, and processes the request.

It is to be noted that in the scenarios described above, the consumer is not aware of the resources behind the *Web Service Provider* or the fact that the service has been dynamically deployed. They interact with the *Web Service Provider* by sending SOAP messages which is the standard way of interacting with a service.



(a) Routing request to existing deployment



(b) Request service not already deployed

Fig. 1. Routing requests in DynaSOAr

DynaSOAr has two other components to support dynamic deployment, namely a *Registry Service*, and the *Service Repository.*

– The *DynaSOAr Registry Service* is provided by GRIMOIRES[7], which is a UDDI-based registry, with added support for storing metadata as RDF triplets. Whenever the provider of a service decides to make the service available via DynaSOAr, the service code needs to be uploaded to the *service repository*, as a result of which the registry is updated with the information. The service is added to the registry without any concrete *accessPoints* (in UDDI terms), but a reference to the *Service Repository* web service is added to it. Every time a service is deployed on any of the available nodes, the entry of that service in the registry is updated with the actual endpoint.
– The *Service Repository* manages the *upload* or *download* of the service code. The Host Providers communicate with this service while downloading the service code for a service to be deployed.

The description so far consisted of a single *Host Provider*. However, in reality, several Host Providers may be available to one *Web Service Provider*. It might be advantageous to make a selection between the available *Host Providers* based on certain parameters, such as cost, dependability, QoS, security. To facilitate this, another component, the broker, with the same interface as the *Host Provider*, has been introduced in the architecture to make such decisions. The broker has the knowledge about one or more *Host Providers*, and is able to make the decisions based on the characteristics of the available *Host Providers* and the QoS or security requirements requested by the consumer. Figure 2 describes the generic architecture of DynaSOAr with all its components.

DynaSOAr is a generic framework allowing the structure to grow dynamically to any level or depth. There can be any number of brokers and any number of *Host Providers*, thereby creating the space for a *Web Service Market Place*, where the brokers can choose between all available providers meeting the consumer requirements to process the requests.

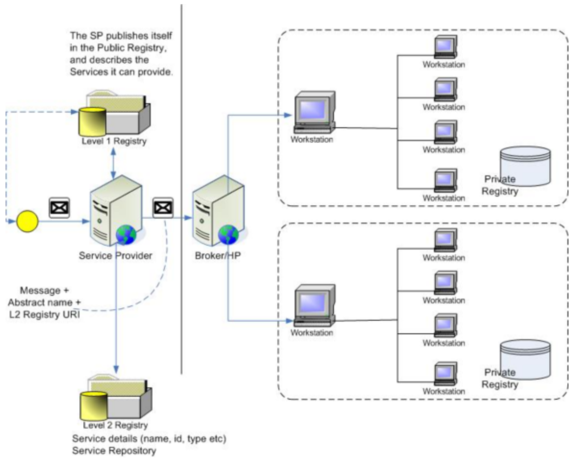

The SP publishes itself in the Public Registry, and describes the Services it can provide.
Level 1 Registry
Service Provider
Broker/HP
Message + Abstract name + L2 Registry URI
Level 2 Registry
Service details (name, id, type etc)
Service Repository
Workstation
Private Registry

Fig. 2. Generic DynaSOAr Architecture

The generic architecture of DynaSOAr does not restrict the dynamic deployment functionality to web services alone. As described in [5], DynaSOAr enables the dynamic deployment of virtual machines like VMWare[8] and Microsoft Virtual PC[9] and also .NET services and stored procedures over SQLServer.

4 Dynamic OGSA-DQP

The features of OGSA-DQP and the requirements for distributed query processing make it a prime candidate for the use of the DynaSOAr infrastructure. Usage scenarios in OGSA-DQP, which can benefit from the DynaSOAr features, include the following -

1. Frequent and long-running queries can benefit from the *on-the-fly* deployment of an analysis service such that it is co-located with relevant data.
2. A new database wrapped by the OGSA-DAI data service can be deployed to enable the GDQS to serve queries involving the new database.
3. An increased degree of parallelism can be obtained by deploying multiple copies of the analysis service on multiple nodes.
4. Increased performance for a database scan can be enabled by deploying virtual machines containing a copy of the database.
5. Even though the deployment of virtual machines is costly in terms of time required, in the case of frequent and long-running use of the database present in the virtual machine or the service deployed in it, the initial deployment cost can be outweighed by the benefits.
6. Polar* performs some basic optimization based on the information available to it. But this optimization can be enhanced by considering the dynamic deployment scenario, where the scheduler should be able to schedule deployment of new evaluation or analysis services on new computational nodes if it finds the existing deployments to be heavily loaded.

4.1 Implementation

As a proof of concept the publicly available OGSA-DQP has been modified to incorporate the DynaSOAr features into it. In the DynaSOAr-enabled OGSA-DQP, the primary requirement is to create a structure similar to the *Web Service Provider - Host Provider* structure of DynaSOAr. The *Grid Distributed Query Service (GDQS)* corresponds to the *Web Service Provider*, which advertises itself as capable of providing distributed query processing functionality over a set of databases exposed as OGSA-DAI data service resources, and a set of analysis services, either provided by a remote provider, or by the GDQS itself. In the latter case, the analysis service may not be deployed on any node available to the GDQS, but the *Service Repository* stores the service code, and the *registry* contains information about this potentially available service.

As in case of the public OGSA-DQP, a *DQP data service resource* must be created from the deployed GDQS factory data service resource when initializing the service. In the second initialization step, this *data service resource* imports the schema of the databases exposed by OGSA-DAI and the WSDL of the analysis service. During this second phase, the GDQS in the dynamic version of OGSA-DQP tries to co-locate the evaluation services with the OGSA-DAI-wrapped databases by dynamically deploying the *Query Evaluation Service*, which is a standard *web service*, onto the nodes where the data resides. If an analysis service advertised by the GDQS is added to the *DQP data service resource* configuration, the GDQS using the DynaSOAr framework deploys the service on a suitable computational node. Once these new services are deployed, the schema and the WSDL are imported from them in the same way as in case of the standard OGSA-DQP. The complete deployment process is shown in Figure 3.

As the new services are deployed, the *registry* is updated with the corresponding information, so that another *data service resource* for the same GDQS will be able to reap the benefit of the previous deployment by re-using the already deployed services. This is the point where the advantages of DynaSOAr over currently available job-based grid systems become apparent. The services deployed using DynaSOAr will be accessible until they are explicitly removed from the server, or the server becomes unavailable, compared to the jobs, which do not persist beyond a single execution. Thus we can achieve a “deploy once, use many times” philosophy with DynaSOAr, which has a positive effect on the performance of the distributed queries as shall be seen in the analysis of the experimental results.

5 Experiment

5.1 Setup

To analyze the performance of the Dynamic OGSA-DQP system, several experiments have been performed and the results analyzed. The initial experiments primarily concentrated on the dynamic deployment of the analysis services and

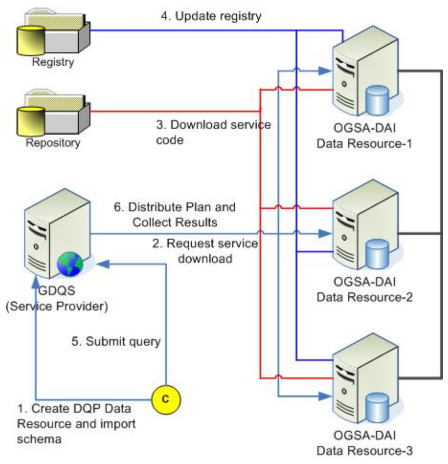


Fig. 3. DynaSOAr enabled OGSA-DQP

the impact made by this on the performance of the distributed query processing activities. The DynaSOAr framework was setup on a set of Linux machines within the Newcastle University GIGA cluster - each of them being a four-processor Intel® Xeon^{TM} CPU 2.80GHz system, with 2GB memory, and Fedora Core 4 installed on them. The GDQS was deployed on another Linux machine - a four-processor Intel® Xeon^{TM} CPU 3.06GHz with 1GB memory and Fedora Core 4 installed on it. The *DynaSOAr Registry* and *Service Repository* service were co-located with the GDQS. The analysis service code, along with the Query Evaluation Service code were uploaded to the *Service Repository*, and a copy of the same analysis service was deployed on a Linux (one-processor Intel® Xeon^{TM} CPU 2.40GHz system with 1GB memory and Red Hat Enterprise 3 Linux) system at the Edinburgh Parallel Computing Centre (EPCC) at Edinburgh University. The network between Newcastle University and EPCC is JANET, which is a high performance gigabit network connecting the universities in the United Kingdom.

Five databases were exposed as OGSA-DAI data resources on five of the Linux systems that were part of the already established DynaSOAr framework. One of the databases used for the test queries was loaded with several tables, each with 100,000 records, and fixed record sizes of 128 bytes, 256 bytes, 512 bytes, 1 Kbytes, 2 Kbytes, 4 Kbytes, 8 Kbytes and 10 Kbytes. The experiments were designed to fetch data out of each table in chunks of 100, 200, 400, 800, 1000, 2000, 5000, 10000, 20000 and 50000 tuples and perform the analysis on each tuple using the analysis service. Results were collected in order to compare the performance of the system with a remote analysis service, to the performance with a local service dynamically deployed using the DynaSOAr framework, i.e to investigate item (1) in Section 4.

5.2 Results and Analysis

In preliminary experiments, the DynaSOAr framework was used to deploy the analysis service on separate hosts. The deployment cost includes the time required to transfer the service code from the repository to the target host and the time taken for the actual deployment within the *web service container*, Apache Tomcat in this case - where the packaged service (packed as a WAR file) is unpacked into a proper directory structure and the various libraries are loaded before the service can be accessed. Figure 4a shows the time taken to deploy the service on different hosts and the average time for deployment.

The average time required for an individual service deployment on a computational node was approximately 32.4 seconds. This is a one-time cost and is incurred only during the DQP initialization phase. Copies of the same service can be deployed in parallel on multiple nodes if required, so that the total deployment cost of all copies becomes equivalent to the cost of a single deployment. Once the service is deployed locally, the performance of the queries executed by this GDQS reaps the full benefit of this one-time deployment, as is evident from the other experiments.

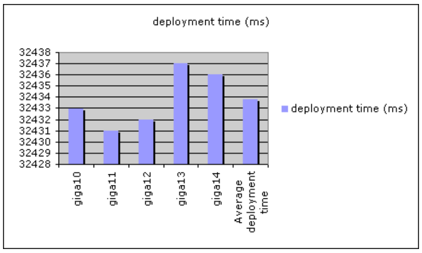
A set of ten queries was executed on a test database, retrieving 100, 200, 400, 800, 1000, 2000, 5000, 10000, 20000 and 50000 tuples from the database. Each query was used to retrieve datasets of different sizes, such as 128 bytes, 256 bytes, 512 bytes, 1 Kbytes, 2 Kbytes, 4 Kbytes, 8 Kbytes and 10 Kbytes. Each query also invoked the analysis service for each retrieved tuple. An example query used in the tests is as follows:

```
%print select p.id, calculateEntropy(p.sequence) from p in
proteinsequence_random_sequence_128s where p.id < 20000;
```

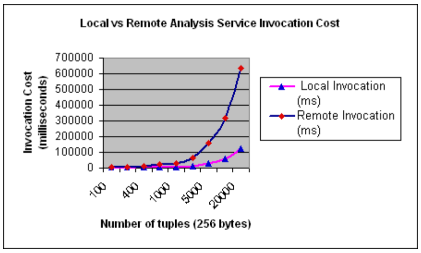
This query retrieves 20,000 tuples from the database and invokes the analysis service (*calculateEntropy* in this case) on the sequence attribute of each tuple. The results of these experiments are shown in Figure 4b and Figure 4c.

Figure 4b compares the invocation cost (in milliseconds) of a local and a remote deployment of the same service for different result cardinalities, ranging from 100 to 20,000. Figure 4c compares the average invocation cost (in milliseconds) of the local and remote service for different sized tuples, from 128 bytes to 10 Kbytes.

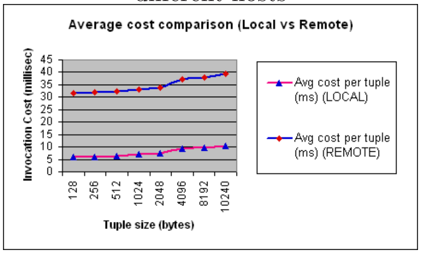
It is evident from the plotted results that the invocation cost increases radically for the remote analysis service as the number of tuples increase starting from 100 tuples to 20000 tuples. In Figure 4b, the total cost of invoking the analysis service increases as the number of tuples retrieved from the database increase. But, the rate of increase is far more substantial when the analysis service is remote, than when it is local. In Figure 4c, the average invocation cost per tuple is plotted against the average tuple size, starting from 128 bytes to 10Kbytes. In this case, for both local and remote services, the invocation cost tends to increase as the tuple size increases, but the effect of a remote service is



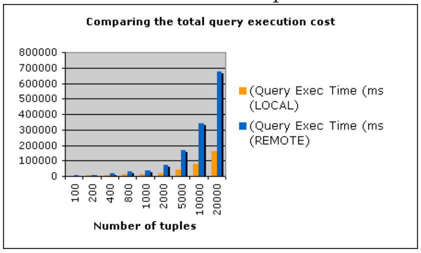
(a) Required deployment time on different hosts



(b) Comparing average invocation cost for different tuple sizes



(c) Comparing local and remote service invocation cost



(d) Comparison of the total evaluation cost

Fig. 4. Performance Analysis of DynaSOAr-enabled OGSA-DQP

significantly higher than a local service, and it can be inferred that the cost of invoking the remote service will increase further if the data size increases.

Figure 4d shows the total query evaluation cost for two scenarios, (1) when the analysis service was local and (2) when the analysis service was remote. This figure shows that the total query evaluation costs when the analysis service was remote are significantly higher than the total evaluation costs when the service was local. The difference between these two values becomes equal to the average cost of deployment (an average of 32.4 seconds) when the number of tuples is approximately 1000, and starts increasing even more significantly as the number of tuples increase. This data validates the statement made earlier in this paper that the one-time deployment cost can be outweighed by the performance benefits in case of frequent, long-running queries.

These results clearly show that for the queries using analysis services over the data retrieved from the databases, the performance of DynaDQP is much better than the standard OGSA-DQP where the analysis service can be remote from the data. The difference in the performance is quite noticeable considering the fact that a very high-speed Internet backbone exists between the server at Edinburgh Parallel Computing Centre and the Linux cluster at Newcastle University. The performance difference would probably be even more prominent if the analysis service resides much further apart geographically, because a higher communication cost would be incurred in that case.

6 Related Work

Although in the DynaSOAr architecture, the Host Provider sits on top of existing Grid infrastructure as a high level service, it can exploit the results of work producing components on which dynamic deployment frameworks can be built. In particular, the job scheduling fabrics like Condor[20] can be utilized as a means of gathering machine characteristics, and CPU loads. However, deployment of services rather than jobs raises other issues, such as making the decision about whether to deploy a service on a new node or to use an existing but possibly overloaded deployment. The GridSHED project [10],[11] for job scheduling has been investigating this area, and the results are being utilized to design an effective scheduling system for DynaDQP.

There is some work on dynamic deployment as in [18], but this is essentially tied to a particular implementation of Grid middleware and web service container (WS-RF[12] and the Globus Toolkit[13]) without addressing the more widespread deployment scenarios involving commonly used standard toolkits such as Axis and Tomcat. The work described in [21] is built on top of specialized hardware. Moreover, the deployment of different types of components such as virtual machines, stored procedures, .NET services together in one framework is not addressed in any of the current systems.

This paper focuses on the use of the dynamic deployment framework within the context of *Distributed Query Processing*. To our knowledge, there is no current distributed query processing system which is factored out as inter-operable services and allows *on-the-fly* deployment of evaluation and analysis services on available nodes thereby co-locating the data processing and analysis code with the data, as proposed in [19]. The analysis of the results clearly indicate that moving the code to the data, even with an initial deployment cost can potentially be beneficial, especially for frequent execution of long running queries over huge data sets.

7 Current and Future Directions

At present, work is under way to enable the DynaSOAr framework (and hence DynaDQP) to support the usage of virtual machines for dynamic deployment of data access services, evaluation services, analysis services and databases. It has been accepted by OGSA-DAI as well that the availability of a deployment-ready OGSA-DAI service would greatly help the dynamic deployment work, and work is going on in that respect too.

In OGSA-DQP, the compiler/optimizer performs some static scheduling based on a very simple cost model, but that does not consider the inherent dynamism in a Grid system where the dynamics of the environment is liable to change drastically. The effects of changes in resources at runtime have been considered in the investigations into *adaptive distributed query processing* [17], [16]. It will be an effective solution to combine the findings of GridSHED, DynaSOAr and the *adaptive DQP* investigation.

Some work has been done in this area of *fault-tolerant distributed query processing* [14], [15]. The concepts of the dynamic DQP are also relevant to fault-tolerant query processing systems where a failure of an evaluation node can be handled through the deployment of the same service on another node or a virtual machine as a replacement of the failed node, and by replaying certain sections of the query evaluation to regain the state where the processing stopped due to the failure.

Virtual Machines are an important aspect for the Dynamic DQP framework. Some basic work has already been done on deploying Microsoft's Virtual PC systems in DynaSOAr. It is being extended to incorporate VMWare systems, and the deployment of databases in virtual machines. Deploying databases in virtual machines does however raise other key issues such as keeping the copy in sync with the original database.

8 Conclusion

This paper presents an overview of ongoing work on enabling dynamic service deployment in OGSA-DQP using the DynaSOAr framework. We believe that distributed query processing can potentially benefit from the dynamic deployment mechanisms of DynaSOAr by deploying evaluation and analysis services closer to the data, and this claim is supported by the experimental results. It also includes scope of creating *software market places*, where the computational resources can be chosen from a pool of available *Host Providers* based on the cost and (or) the quality of service provided by the host.

The project is continuing to investigate different aspects of the system, such as scheduling new deployments, routing requests between multiple instances of the same service, deploying virtual machines and databases, and the future work involves looking into the utility of the framework for adaptive and fault-tolerant distributed query processing systems and evaluating various transport technologies for transferring the service code.

Acknowledgments. We wish to extend our gratitude to our collaborators in GridSHED: Isi Mitrani and Jennie Palmer. We wish to acknowledge the support from our colleagues in OGSA-DAI, OGSA-DQP and DAIT projects at Manchester University and EPCC. We wish to thank Jim Smith for his helpful discussions. We would also like to thank the UK Engineering and Physical Sciences Research Council, and the DTI for the GridSHED and DAIT projects.

References

1. Alpdemir, M.N., Mukherjee, A., Gounaris, A. et.al.: OGSA-DQP: A Service for Distributed Querying on the Grid. In: Advances in Database Technology - EDBT 2004. Lecture Notes in Computer Science, Vol. 2992. Springer-Verlag, 858–861
2. OGSA-DQP, `http://www.ogsadai.org.uk/about/ogsa-dqp/`
3. OGSA-DAI, `http://www.ogsadai.org.uk/`

4. OGSA-DAI Glossary of Terms, http://www.ogsadai.org.uk/documentation/ogsadai-wsrf-2.2/doc/reference/glossary.html
5. Watson, P., Fowler, C., Kubicek, C., Mukherjee, A. et. al.: Dynamically Deploying Web Services on a Grid using Dynasoar. In: Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, ISORC 2006, IEEE Computer Society 2006
6. Smith, J., Gounaris, A., Watson, P. et. al.: Distributed Query Processing on the Grid. In: Grid Computing 2002. Lecture Notes in Computer Science, Vol. 2536. Springer-Verlag, 279–290
7. GRIMOIRES, http://twiki.grimoires.org/bin/view/Grimoires/
8. VMWare, http://www.vmware.com/
9. Microsoft Virtual PC, http://www.microsoft.com/windows/virtualpc/default.mspx/
10. Palmer, J., Mitrani, I.: Optimal Server Allocation in Reconfigurable Clusters with Multiple Job Types. In: Computational Science and its Applications (ICCSA 2004), Assisi, Italy, 2004.
11. Kubicek, C., Fisher, M., McKee, P., Smith, R.: Dynamic Allocation of Servers to Jobs in a Grid Hosting Environment. BT Technology Journal, Vol. 22, 2004. 251–260
12. Web Services - Resource Framework, http://www.globus.org/wsrf
13. Globus Toolkit, http://www.globus.org/toolkit
14. Smith, J., Watson, P.: Fault-Tolerance in Distributed Query Processing. In: 9th International Database Engineering And Application Symposium. IDEAS 2005, http://ideas.concordia.ca/ideas2005/, IEEE, 329–338
15. Smith, J., Watson, P.: Failure Recovery Alternatives In Grid Based Distributed Query Processing: A Case Study. The University of Newcastle upon Tyne, number CS-TR-957, April 2006
16. Gounaris, A., Paton, N.W., Sakellariou, R., Fernandes, A.A.A. et. al.: Practical Adaptation to Changing Resources in Grid Query Processing. In: The 22nd International Conference on Data Engineering, ICDE 2006
17. Gounaris, A., Paton, N.W., Sakellariou, R., Fernandes, A.A.A. et. al.: Adapting to Changing Resource Performance in Grid Query Processing. In: VLDB Workshop on Data Management in Grids, DMG 2005, http://liris.cnrs.fr/~jpierson/DMG_VLDB05/
18. Qi, L., Jin, H., Foster, I., Gawor, J.: HAND: Highly Available Dynamic Deployment Infrastructure for Globus Toolkit 4, http://www.globus.org/alliance/publications/papers.php#HAND
19. Watson, P., Lee, P.: The NU-Grid Persistent Object Computation Server. In: 1st European Grid Workshop, Poznan, Poland, 2000
20. Tannenbaum, T., Wright, D., Miller, K., and Livny, M.:Condor - A Distributed Job Scheduler. In:Beowulf Cluster Computing with Linux, T. Sterling, Ed.: The MIT Press, 2002.
21. Chrysoulas, C., Haleplidis, E., et. al.:Applying a Web-Services Based Model to Dynamic-Service Deployment. In: International Conference on Intelligent Agents, Web Technology, and Internet Commerce (IAWTIC), Vienna, Austria, November 2005