Performance evaluation of a grid resource monitoring and discovery service

H.N. Lim Choi Keung, J.R.D. Dyson, S.A. Jarvis and G.R. Nudd

Abstract: The Grid Information Service (GIS) is one of the Grid Common Services which make up the basic functions belonging to Grids. This service offers a resource discovery mechanism, of which an implementation is provided by the Monitoring and Discovery Service (MDS-2), which is part of the Globus Toolkit®. Grid applications are typically the users of this resource discovery mechanism; knowledge is thus obtained about the existence of appropriate resources on which to execute jobs. Since these Grid applications are likely to be influenced by dynamic middleware, it is crucial that they experience a reliable performance for the proper utilisation of resources and for accountability purposes. Consequently, the approach taken in the paper is to investigate and evaluate a level of performance obtainable from the MDS. A better understanding of the performance of the GRIS (Grid Resource Information Service) and subsequently the GIIS (Grid Index Information Service) leads to a more standard way of formalising the performance of a Grid Information Service. This is achieved by defining a number of performance metrics which are analysed against a set of different information gathering methods and GRIS back-end implementations.

1 Introduction

The Grid [1] provides a platform where the scientific and engineering communities can share data and computation across multiple administrative domains. There are several key services that must be offered by Grid middleware; one of them is the Grid Information Service.

A Grid Information Service (GIS) is a Grid middleware component which maintains information about hardware, software, services and people participating in a virtual organisation (VO) [2]. On request from Grid entities, the GIS can launch resource discovery and lookup [3]. Several solutions exist which implement such information services: Globus, Legion, Condor and Ninf [4–7], amongst others. This paper investigates the Monitoring and Discovery Service (MDS-2) [8], the Grid Information Service provided by the Globus Toolkit.

Performance is likely to be an important issue for Grid applications running on top of middleware services. For example, the overall performance of an application will be partly dictated by the performance incurred using a resource discovery service. This performance issue is further complicated by the resources being heterogeneous and by the Grid fabric having variable bandwidth and latencies. Such a dynamic and heterogeneous environment could result in an unpredictable and unreliable outcome for applications. As a result, this situation hinders the forecasted scheduling of resources and the accountability of applications.

Previous work has studied the performance of the MDS when authenticated queries are issued and when anonymous ones are used [9]. Our contribution also complements a recent paper [10] where the difference in scalability obtained from three monitoring and information systems: MDS2.1, R-GMA and Condor's Hawkeye, is examined. Scalability results are obtained and analysed when various system components are subjected to increasing user loads. Our work is different from [10] in that we focus on the difference or similarity in querying an aggregate information server and an information server for the same set of information, based on the varying information gathering methods at the source. Moreover, the contribution which we make addresses comparable issues as in [10] – for instance, the effect network behaviour has on the performance observed, especially, the load of the information server. All these performance studies are crucial in proposing recommendations for the deployment of the monitoring and information systems under consideration. The contribution of this paper is therefore an analysis of the performance obtainable from the GRIS and the effect on Grid applications. More specifically, this paper studies the way in which data are provided by information providers to the GRIS. The latter advertises Grid resource information about a node [11] from multiple information sources. Depending on whether caching is enabled in the GRIS, it is possible that all the information providers are executed for a query. Alternatively, a periodic information update mechanism can be implemented. These various methods affect the query performance in different ways; therefore, in this paper, a GRIS will be repeatedly queried and the performance obtained will be analysed.

2 The monitoring and discovery service (MDS)

The functionality of the Globus MDS is encapsulated in two components: the Grid Resource Information Service (GRIS) and the Grid Index Information Service (GIIS) [8]. The GRIS runs on Grid resources and is an information provider framework for specific information sources. On the other hand, the GIIS is a user accessible directory server that accepts information from *child* GIIS and GRIS instances and aggregates this information into a unified space. It also supports searching for resources by characteristics.

IEE Proceedings online no. 20030809

doi: 10.1049/ip-sen:20030809

Paper received 25th April 2003

The authors are with the High Performance Systems Group, Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK

There are a number of ways by which an information provider can supply information to a GRIS. These are explained below.

2.1 Evaluation methods

The definitions of the meanings of the different evaluation methods [12] are:

1. *Lazy evaluation*: Obtain freshly generated information on receiving a search request.
2. *Eager evaluation*: Obtain freshly generated information on receiving the first search request, and cache it in the GRIS. Thereafter, check the cache to see if the subsequent search requests can be serviced. If the cache TTL (time-to-live) has not been reached, then the requests are serviced out of the cache. Otherwise, obtain and cache freshly generated information (lazy evaluation).
3. *Speculative evaluation*: Information is generated frequently and placed in a recognised location. On receiving a search request, service is provided from information in that location. There is no caching in the GRIS in this method. Here, the information being returned for the search request may not be fresh as in the lazy evaluation method, but it is readily available and is frequently updated.

3 Performance of the MDS

3.1 Introduction to performance of the MDS

Performance is an issue for Grid applications as they execute on heterogeneous resources with variable bandwidths and latencies [3]. Since it is difficult to characterise the performance of such a dynamic and heterogeneous environment, it is increasingly important to provide a reliable performance through quality-of-service.

The performance of a query to the MDS cannot be predicted with a formula. This is because the predictability of the performance decreases with the complexity of the GIIS hierarchy. Moreover, the length of time a query might take to answer depends on the time-to-live (TTL) data [13]. The approach normally taken to counter this performance variability is to ascertain that the data being requested are in cache. When queries are sent to the MDS, they will be serviced by data in the cache, which is regularly refreshed. Another method is to increase the TTL value for the data, though this might not be feasible for dynamic data. These techniques could be applied to any GRIS or GIIS independently or at the same time.

Due to the inconsistencies above, it is crucial to perform tests at the smallest unit that can exist in the MDS hierarchy, namely the GRIS. Understanding how information is provided at the lowest level is important because performance is unpredictable and depends on the complexity of the hierarchy. Furthermore, the number of information providers is large, and as they are the original sources of information, they are accessed frequently. These factors affect the performance with which this information is propagated upwards to the higher level GIIS nodes.

3.2 Assessment of performance

Middleware has a number of characteristics: size, cost, complexity, flexibility and performance. The importance of each characteristic depends on the application. One way of assessing performance is to measure the rate at which requests can be sent and results received through a given system. The most performant middleware system is the one with the least time taken for the messages to pass through it. When time measurements are taken from the sending moment to the receiving moment, no application-specific setup is considered. The measurement thus effectively represents the behaviour of the middleware and the network and operating system below it.

The difference must be made between *high performance* and *high quality* middleware. Middleware performance can be measured by the time taken to carry out an operation. However, the fastest middleware is not the only issue for many middleware applications. Other issues, including scalability, flexibility and adaptability, ease of use, tool support and standards conformance could be more important, depending on the application. Nevertheless, these characteristics are very difficult to measure since they are relatively subjective.

Furthermore, scalability depends on performance; for example, for an information service to handle a large number of concurrent requests, it needs to process each request at a high speed. If the information service takes a shorter time to process one request, it can handle more requests in a particular period of time.

However, high performance alone does not guarantee high scalability. For instance, it is possible that some information services perform well when run single-threaded but as soon as multiple threads of control are used, the performance greatly degrades. This occurs because all the threads are competing for the same resources and so must block and waste cycles, attempting to acquire the locks required to use the resources.

In the experiments in this paper, reliable performance is built on an analysis of the performance of the Grid Information Service. Since it is important to analyse the GRIS, this paper investigates the different information provider mechanisms and the benefits of using different caching strategies.

4 Experimental environment

The experiments were carried out on a Grid testbed at the University of Warwick and were based on MDS 2.1. This particular version of the MDS was chosen because MDS 2.x is currently utilized in the majority of UK e-Science projects [14] and US testbeds, including NASA's Information Power Grid [15]. Across the various experiments, the following agent setup was maintained. Agents make request queries to the MDS, which are sent from a set of ten machines (*mscs-02* to *mscs-11*). With a maximum of 500 agents simultaneously making queries over a period of 10 min, the desired effect was to load-balance the queries and to sustain the MDS querying. The maximum number of agents attributed to one machine is therefore 50 [Note 1]. The time it takes for each request to be serviced is measured and an average response time is calculated. Moreover, every agent sleeps for 1 s before sending the next request.

To test the scalability of the MDS, a GRIS was set up [Note 2]. For the lazy evaluation experiment, the MDS default cache time values were used. Furthermore, for the experiments, the core information providers included by default in the MDS are used. These include the GRAM (Globus resource allocation manager) reporter, which

Note 1: The *mscs* machines each have the Linux operating system with kernel 2.4.18-27.7.x, a 2.4 GHz processor and 512 MB RAM. The *mscs* machines are also on an Ethernet LAN and they are connected to the GRIS host by a 100 Mb link.

Note 2: It was set up on a Linux kernel 2.4.18-14 machine (M_1), which has a 1.9 GHz processor and 512 MB RAM.

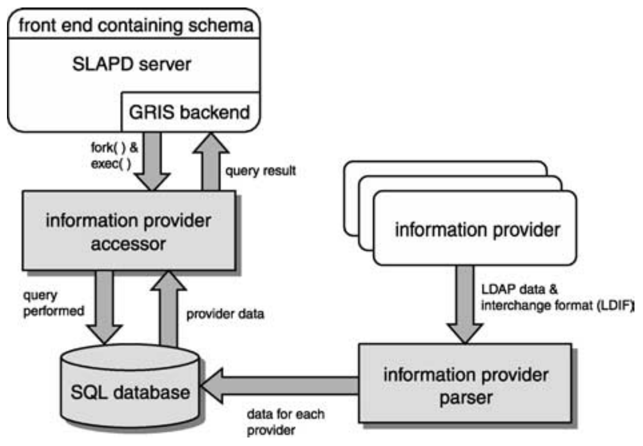

Fig. 1 *Components of the system*

provides information about the *fork* job manager. The aim of the experiments was to analyse the performance of the GRIS with a minimal number of information providers which offer relevant status information about the Grid.

To evaluate the performance of the GRIS, an agent application was developed to query either a GRIS or a GIIS. One set of experiments was carried out with the agent using the Java CoG Kit [16] libraries and the other set with the Globus C APIs. The aim was to determine whether the implementation language actually affected the performance obtained when querying the MDS. The Globus C and Java CoG APIs have been used instead of visualisation tools because it is typical for Grid applications to use those APIs directly.

Figure 1 shows the different components of the experimental setup with a focus on speculative evaluation (SE).

4.1 Experimental factors

The approach in this paper is to consider that performance is defined by a number of factors, including response time, throughput, the total number of responses and system load. The scalability of the MDS directly depends on the performance observed.

The average response time in seconds is the mean time it took each query made by an agent, to be serviced by the respective MDS component. The average throughput denotes the mean number of queries which are processed by the GRIS per second. Each experiment also keeps a count of the number of successful queries processed by the MDS. Moreover, the system load has been measured as the load on the system during the last minute (1 min load average) and during the last 5 min (5 min load average). The load average is a measure of the number of jobs waiting in the run queue.

In this paper, the complexity of queries is not assessed. Therefore, the complex relationships amongst data objects such as the sophisticated searches on objects are not being tested. The experiments only query the MDS for all the data available; Grid applications are likely to carry out such an operation to discover the status of the Grid. Thus, queries are sent to the MDS by specifying the *subtree* search scope which searches from the base DN (distinguished name) [17] and down. Once the data are returned from the MDS, filtering could happen as a subsequent step.

Up to 500 agents queried the MDS simultaneously; they also waited 1 s when they received a query response, before issuing the following request. In the speculative evaluation method using a relational database, the frequency with which the information providers wrote to the database simulated the cache TTL of the core information providers. The performance measurements obtained are:

- average response time in seconds ($\mathcal{R}_T$)
- average throughput in terms of the number of queries answered per second ($\mathcal{T}$)
- total number of successful query responses ($\mathcal{R}_e$)
- 1 min load average ($\mathcal{L}_1$)
- 5 min load average ($\mathcal{L}_5$).

5 Experiment contexts and results

The behaviour of the MDS can only be understood after the collection of performance data for a long period of time. Therefore, the experiments were carried out five times and an average taken.

The experiment results graphs show the situations below.

5.1 GRIS back-end implementations

The curves in all experiments are:

- ***Lazy evaluation (LE)***: The GRIS cache TTL is equal to zero. On receipt of each query, the GRIS launches its core information providers.
- ***Eager evaluation (EE)***: The GRIS cache TTL is not equal to zero (default values) and the information providers are invoked when the cache is expired. The cache is also filled with the new data.
- ***Java speculative evaluation (SE) (PostgreSQL) [Note 3]***: The GRIS cache TTL is equal to zero and the information providers write their data to a relational database (PostgreSQL) on average every minute. The frequency at which these information providers write to the database is set to emulate the GRIS cache TTL. The information provider accessors are written in Java.
- ***Java speculative evaluation (MySQL) [Note 4]***: The GRIS cache TTL is equal to zero and the information providers write their data to a MySQL database on average every minute. The frequency at which these information providers write to the database is set to emulate the GRIS cache TTL. The information provider accessors are written in Java.
- ***Eager & Java speculative evaluations (PostgreSQL)***: The GRIS cache TTL is not equal to zero and the information providers write their data to a relational database (PostgreSQL) on average every minute. The information provider accessors are written in Java.
- ***Eager and Java speculative evaluations (MySQL)***: The GRIS cache TTL is not equal to zero and the information providers write their data to a MySQL database on average every minute. The information provider accessors are written in Java.
- ***Perl speculative evaluation (PostgreSQL)***: The GRIS cache TTL is equal to zero and the information providers write their data to a relational database (PostgreSQL) on average every minute. This implementation is similar to the Java speculative evaluation method, with the difference lying in the information provider accessors which are written in Perl.
- ***Perl speculative evaluation (MySQL)***: The GRIS cache TTL is equal to zero and the information providers write their data to a MySQL database on average every minute. The information provider accessors are written in Perl.

Note 3: The version of PostgreSQL used is 7.2.3 with no extra features such as caching enabled.

Note 4: The version of MySQL used is 4.0.1 and caching is disabled.

In all of the speculative evaluation methods, the information providers themselves are written in Perl rather than Java due to their shorter execution time.

The experiments have also been repeated using C agents in order to determine whether the implementation language of the application querying the MDS makes a difference to the overall performance. It was found that there was no major difference between Java and C agents. Thus, Java agents will be used to show the results obtained for measuring only the performance of the MDS and not that of the applications.

Additionally, experiments have shown that there is no marked difference between the performance of an agent written in Java CoG and one written in JNDI (Java Naming and Directory Interface) [18]. Therefore, all the experiments will involve Java CoG.

5.2 Experiment: GRIS scalability with Java CoG agents

This experiment tests the way in which the scalability of a GRIS changes with an increase in the number of Java CoG agents. Moreover, the eight GRIS back-end implementations discussed in Section 5.1 have been implemented, and the results obtained are shown in Figs. 2–4.

From Fig. 2 it is seen that the average response time generally increases with up to 150 concurrent agents but it slowly stabilises when that number of agents increases. It has been found that the best average response time consistently results from the EE implementation. The second best average response time is obtained with the EE and Java SE; in both cases, the response time does not increase beyond 3.6 s. These behaviours can be explained by the fact that caching data in the GRIS reduces the length of time it takes to process a query. Furthermore, using the Java SE method with GRIS caching only increases the response time by an average of 49%. The benefit of using the Java SE with GRIS caching is to have immediate access to data which is updated periodically, as compared to launching the information providers when there is a cache miss.

The lazy and Perl SE methods have a similar average response time with an increase in the number of concurrent agents. However, the Perl SE has a slightly better response time. The Java SE implementation produces the most significant increase in average response time with an increasing number of agents. This is due to the overhead caused by running the Java Virtual Machine (JVM) for each query. Furthermore, the Java SE methods using GRIS caching and both a PostgreSQL database and a MySQL database produce similar behaviour. This can be explained by the fact that, on average, for each query, the data required are in cache and, therefore, accessing the database does not affect the average response time from the GRIS. Nevertheless, it can be seen that once GRIS caching is turned off, the Java SE consistently performs better with MySQL than with PostgreSQL. In contrast, if PostgreSQL is used with Perl information provider accessors, then this performance is similar to the Java SE with MySQL. Therefore, it can be deduced that using Perl scripts where possible can dramatically reduce the average response time by 65%.

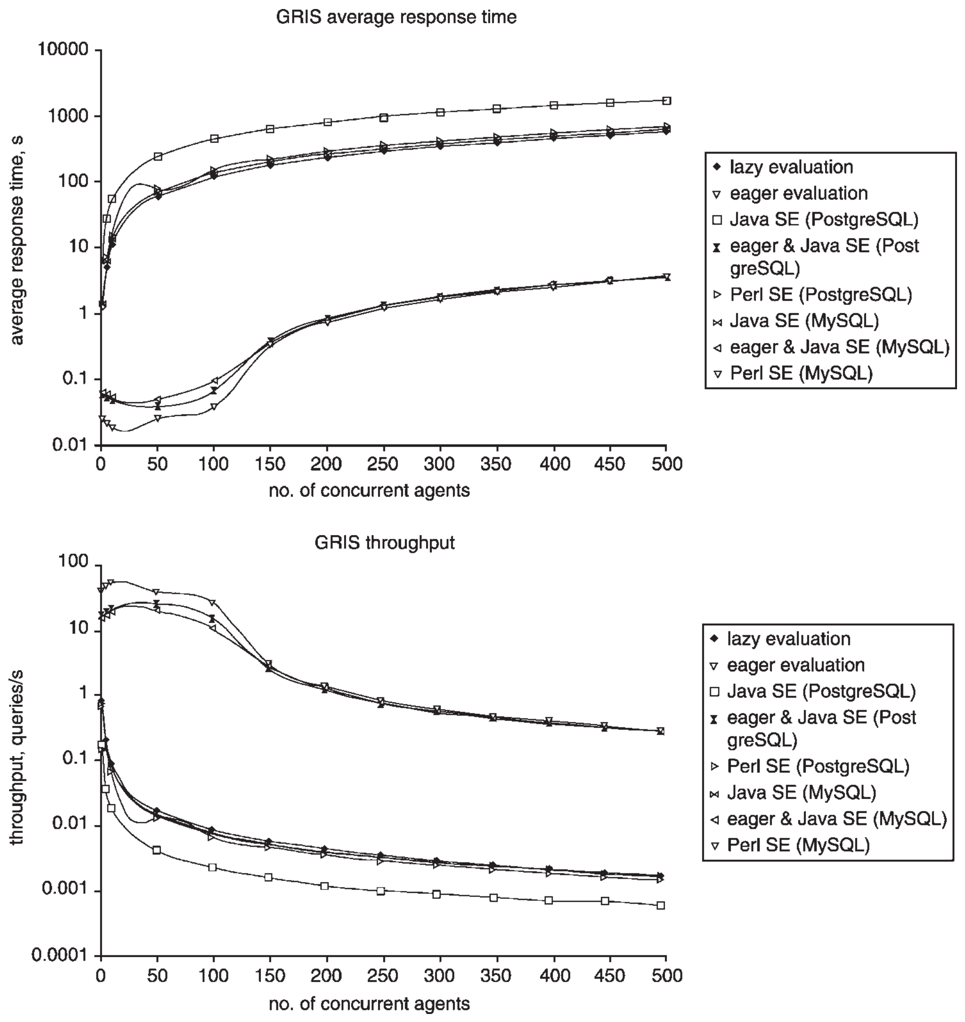

Fig. 2 *Experiment average response time and throughput*

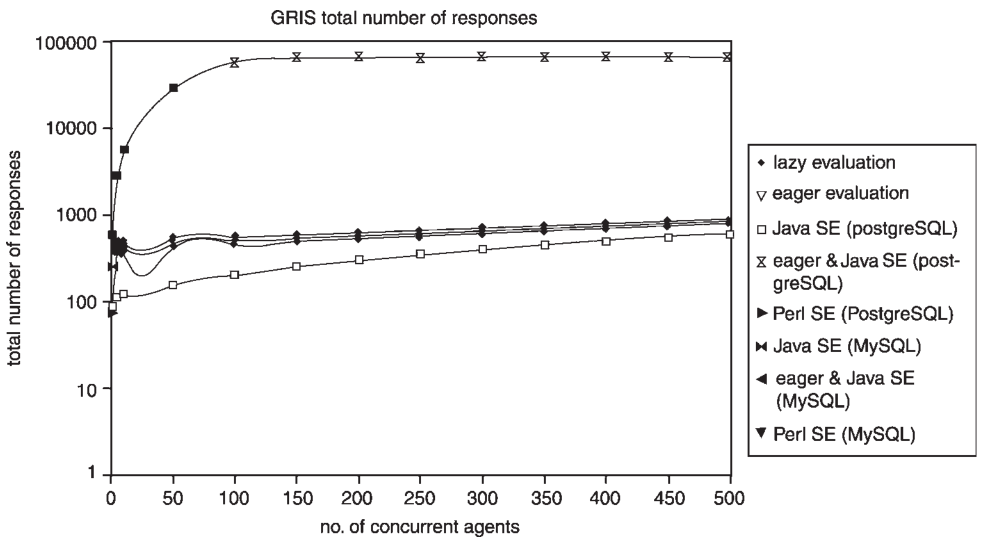

Fig. 3 *Experiment total number of responses*

On the bottom graph of Fig. 2, the throughput achieved by the MDS is shown. The relationship between throughput and response time is

$$\mathcal{R}_T \propto 1/\mathcal{T} \tag{1}$$

Since $\mathcal{T}$ is inversely proportional to $\mathcal{R}_T$, the number of queries processed per second is the largest with EE and the smallest with Java SE (PostgreSQL). Caching in the GRIS also causes the average throughput to increase from 1 to 50 agents and then to sharply drop when the number of agents increases to 150. This effect shows that caching can make the GRIS more efficient in handling an increasing number of

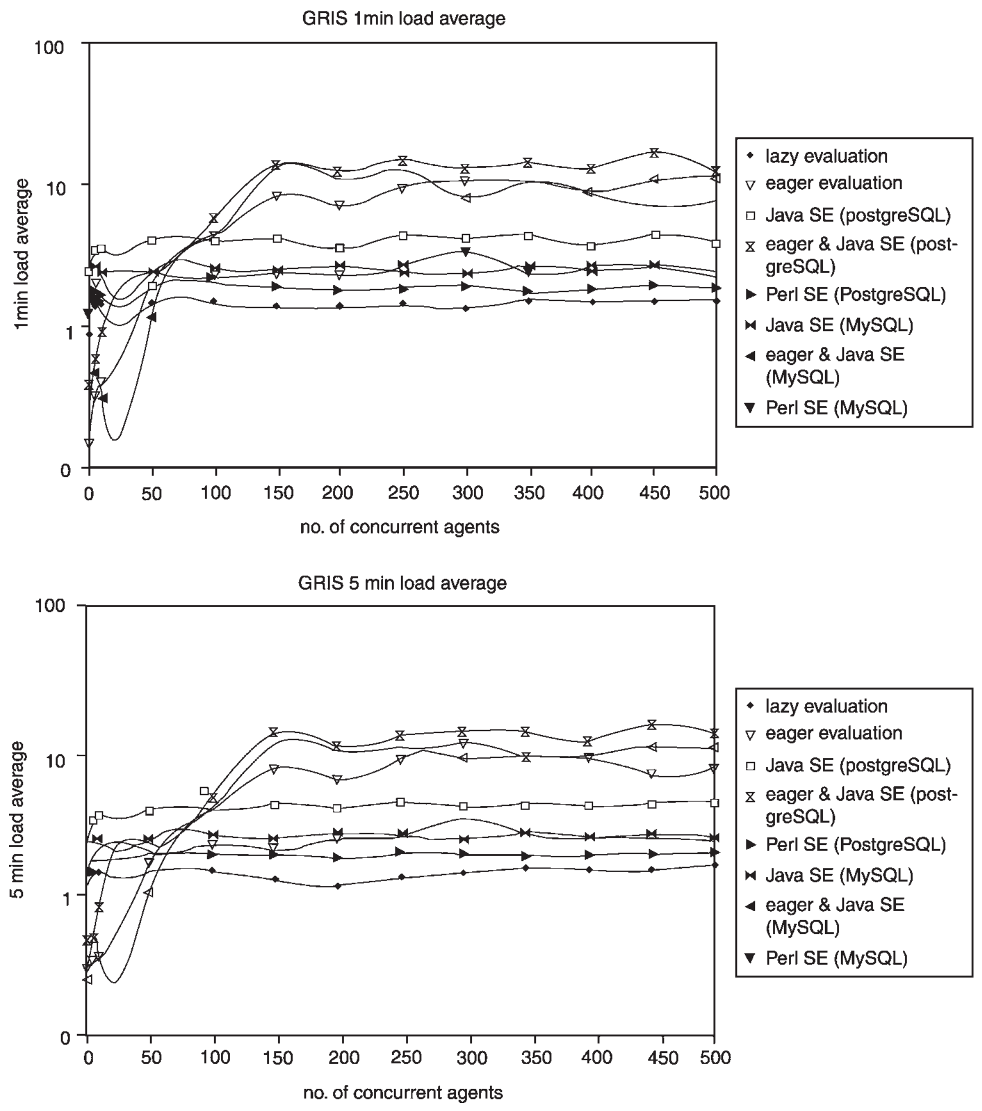

Fig. 4 *Experiment load averages*

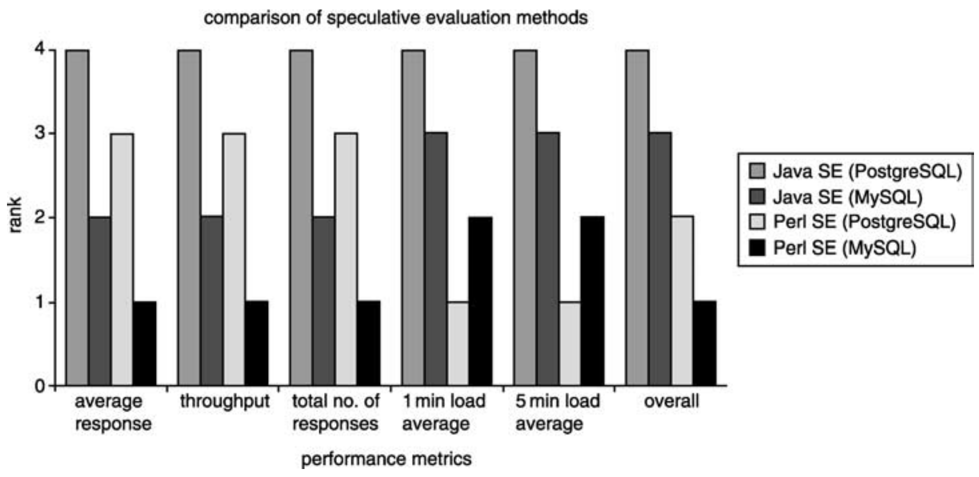

Fig. 5 *Performance analysis of different speculative evaluation methods*

queries up to a certain point. The throughput with Perl SE (MySQL) is only very slightly higher than that for Java SE (MySQL); this occurs for up to 150 agents, after which both throughputs are the same. The slightly better efficiency of Perl over Java levels off for more than 150 agents because the GRIS has reached its maximum query processing capability.

The graph in Fig. 3 shows the total number of successful query responses, which return as the number of agents simultaneously querying the GRIS increases. The three curves involving EE display a gentle increase in the number of responses (about 5000) when up to 10 agents concurrently query the GRIS. Then, there is a sharp, steady increase in $\mathcal{R}_e$ when the number of agents increases up to 150. As this number continues to increase, $\mathcal{R}_e$ stabilises at around 65 000. Again, caching in the GRIS enables more queries to be serviced; however, for more than 150 agents, the GRIS has reached its saturation point and it cannot process more than about 66 000 queries.

The rest of the curves, apart from Java SE with PostgreSQL, show comparable behaviour with $\mathcal{R}_e$ ranging from 76 to 1000. Because the average response time for Java SE (PostgreSQL) was significant, its $\mathcal{R}_e$ struggles to increase beyond 600 for 500 agents.

Figure 4 shows the load averages on the GRIS node for 1 min and 5 min. Querying the GRIS using EE places an increasing load on the GRIS with the number of agents increasing up to 150. This behaviour can be explained by the sharp increase in $\mathcal{R}_e$ seen in the previous graph. But for any further increase in the number of agents, the load remains stable at around 10.0 due to the GRIS being unable to process a larger number of queries. The remaining curves show a relatively stable load with an average of 2.0 and not exceeding 4.4. Moreover, MySQL places a lower load on the GRIS node than PostgreSQL, with Java as the information provider accessors; however, the inverse is true for Perl accessors. The loads shown with MySQL follow each other very closely indeed.

5.3 Speculative evaluation analysis

Since the four different speculative evaluation methods display contrasted performance measurements, they are analysed in Fig. 5 to show how they compare.

In the Figure, the *y*-axis (Rank) denotes how well each SE method fares. The lower the rank, the more performant the method. The *Overall* label indicates the overall rank for each of the methods; the best performance is achieved by Perl SE with MySQL, and the worst is Java SE with PostgreSQL.

5.4 Speculative evaluation with GRIS caching

As well as using the above SE methods, GRIS caching can be enabled. The caching effect is illustrated in the bar chart, Fig. 6. This shows the averages for each performance

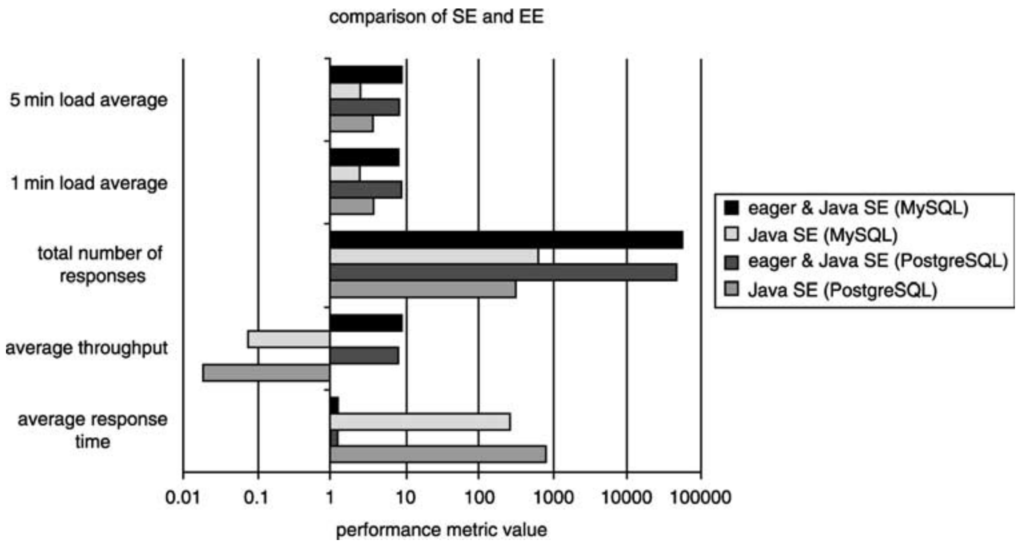

Fig. 6 *Performance analysis of different SE and EE methods*

metric; while GRIS caching reduces the average response time, it increases the average throughput, total number of responses and load averages. Moreover, comparing EE and SE (PostgreSQL), and EE and SE (MySQL), shows that caching leads to the performance metric values being very similar.

6 Performance analysis of the GRIS

These experiments show that different GRIS back-end implementations can affect the performance perceived by the entity querying it. The cache TTL and the timeout on the GRIS have been left constant to analyse the efficiency of caching and various information gathering methods. For instance, caching information in the GRIS drastically reduces the average response time for a query, but the load placed on the GRIS is relatively high. Having a GIIS on the same node as the GRIS might increase the load further and deteriorate the average response time and the number of successful queries. Moreover, if a minimal average response time is required from the GRIS, lazy evaluation should not be used, unless it is required that the load on the GRIS node be at its minimum. The lowest load during the experiment was with LE, and it stayed relatively constant at approximately 1.5.

The advantage of SE is to facilitate the information updating mechanism which happens periodically as opposed to when there is a cache miss. SE also ensures that data which is still within its TTL is readily available in a database. Nevertheless, SE with both MySQL and PostgreSQL are slightly less performant than LE. Combining EE and SE results in an almost similar average response time and number of responses to EE.

Choosing the method by which an information provider will provide information depends on the forecasted number of agents that will query the GRIS at the same time. The experiments in this paper showed the worst case scenario whereby a number of agents kept querying the GRIS over a period of 10 min. If less than approximately 45 agents are querying the GRIS at any one time, it would be preferable to use the methods involving EE because their loads and $\mathcal{R}_T$ on the GRIS node are at their lowest. The cost of these methods, however, is data which might be out-of-date. Therefore, the method chosen depends on both the simultaneous number of queries and the permissible margin of information staleness. For example, information which rarely changes, like the operating system, has a greater margin than dynamic information. For more than 45 concurrent agents, the load with the EE methods increases dramatically and stays stable at a level much higher than the other methods, even SE. Again, there is a trade-off amongst $\mathcal{R}_T$, GRIS load and information freshness. If the agent requests the most up-to-date information, then LE should be used, but at a higher cost of larger $\mathcal{R}_T$, than EE.

The choice of implementation language for the information providers and the database in the SE methods also affects performance. Perl generally provides a smaller $\mathcal{R}_T$ than Java, and MySQL is more performant that PostgreSQL.

7 Performance evaluation of the GIIS

The performance perceived by a client, for example an agent, when it queries a GIIS directly depends on the performance of the GRISes which are registered with it. The performance is also dependent on the time-to-live value of the information the GRISes return to the GIIS. Usually, a quick response can be expected when the data are cached. Nevertheless, when the requested data have expired in the cache, the GIIS will query the lower-level GRISes which supply the data. The performance of this sub-query is equivalent to a client directly querying the GRIS, an analysis of which is given in this paper.

Since the observable performance of a query to a GIIS relies on the complexity of the GIIS hierarchy, it is important to analyse and understand the performance of a

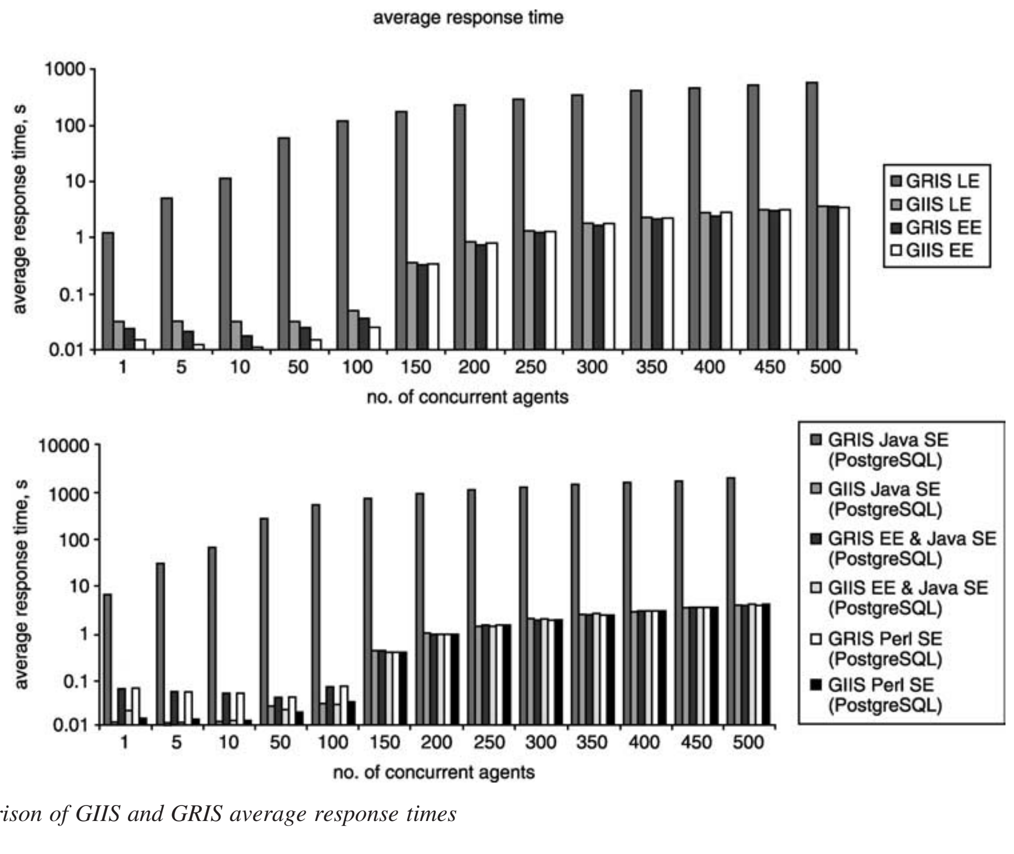

Fig. 7 *Comparison of GIIS and GRIS average response times*

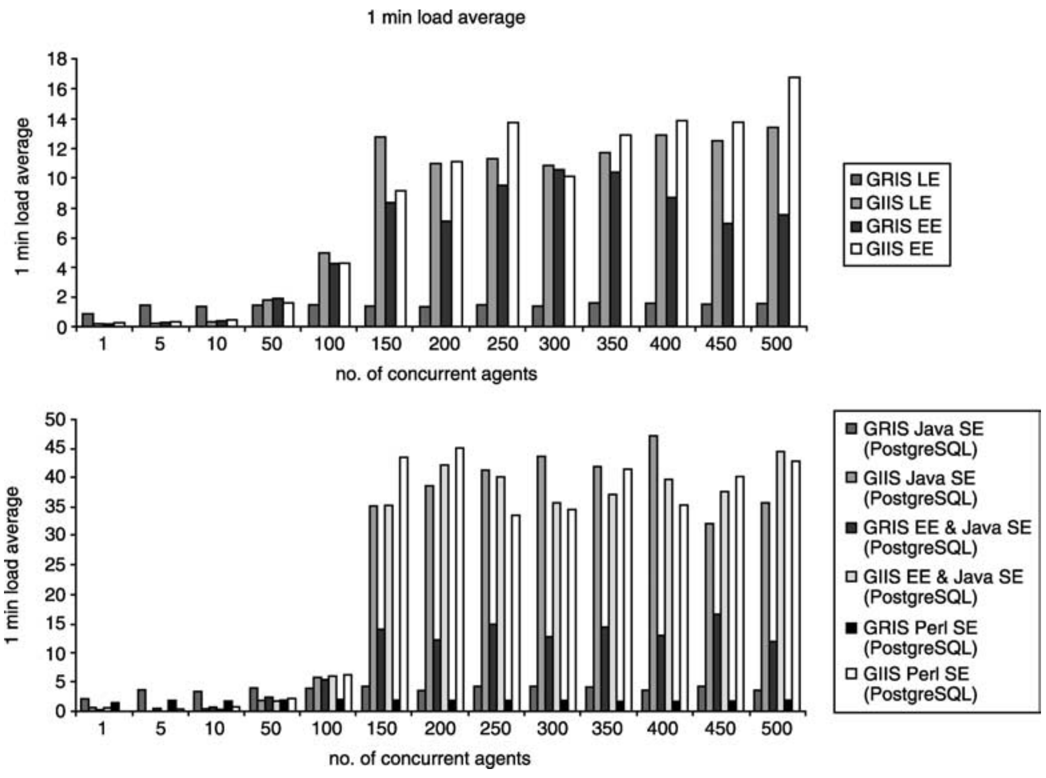

Fig. 8 *Comparison of GIIS and GRIS 1 min load average*

simple GIIS hierarchy. Additional experiments have been carried out to investigate the resulting performance when a user-level application queries a GIIS which has a single GRIS registered to it. Both the GIIS and the GRIS are on the same server machine.

The experiments for the GIIS are similar to those for the GRIS, except that agents query the GIIS instead of the GRIS. More specifically, the experiments observe the change in scalability of an aggregate server with an increase in the number of agents. It has been found that when the GIIS cache TTL is set to zero and the GRIS works by lazy evaluation, poor performance is obtained from the GIIS for more than 10 concurrent agents. Similarly, when the GIIS cache TTL is increased to 60 s, the GIIS performance declines after more than 100 concurrent agents. Increasing the GIIS cache TTL further to 1800 s allows up to 200 agents to concurrently query the GIIS with adequate performance. Therefore, to obtain an acceptable level of performance from the GIIS, its cache TTL has been set to 3600 s throughout these experiments.

7.1 Comparison of GIIS and GRIS performance

Figure 7 shows the differences in the average response times of a GRIS and a GIIS with increasing concurrent requests. GIIS LE is more efficient than GRIS LE throughout the whole of the experiment. The GIIS $\mathcal{R}_T$ remains relatively constant as the number of agents querying the GIIS increases from 1 to 100, but there is a sharp increase in $\mathcal{R}_T$ thereafter. With EE, the GIIS $\mathcal{R}_T$ is only slightly better than that for the GRIS as the number of agents increases from 1 to 100. However, as this number increases to 500, the GRIS and GIIS $\mathcal{R}_T$ are very similar. These observations can be explained by the caching effect in both the GRIS and the GIIS. Similar results are seen with Java SE (PostgreSQL), where $\mathcal{R}_T$ stays relatively constant as the number of agents increases to about 100. Then, there is an increase to around 3.5 s at 500 agents. The GRIS and GIIS $\mathcal{R}_T$ for EE and Java SE (PostgreSQL) closely follow that for EE. Moreover, the Perl SE experiments show that $\mathcal{R}_T$ is lower than for EE and Java SE (PostgreSQL). However, it is similar to that for Java SE (PostgreSQL), indicating that caching in the GIIS levels out any differences in $\mathcal{R}_T$ from the GRIS.

The differences in 1 min load average between a GRIS and a GIIS are shown in Fig. 8. The GIIS 1 min load average with LE steadily increases as the number of agents increases from 1 to about 150, and thereafter it stabilises at around 13.0. When the number of agents is less than 50, the load with only the GRIS is higher than that with both the GRIS and the GIIS. However, as the number of agents keeps increasing, $\mathcal{L}_1$ for the GIIS becomes greater than that for the GRIS. Similar results are seen for LE $\mathcal{L}_5$. With less than 50 agents, caching in the GIIS allows the load on the node to be small but a larger number of agents causes the load to increase as more information providers have to be executed. In EE, $\mathcal{L}_1$ for both the GRIS and the GIIS closely follow each other; the same results occur with $\mathcal{L}_5$. For Java SE, similar results are obtained as for LE, with the difference that the load stabilises at about 40.0; SE places a much higher load on the node. In addition, Perl SE places a slightly higher load on the node, and EE and Java SE produce higher $\mathcal{L}_1$ and $\mathcal{L}_5$ due to the overhead incurred by SE.

8 Ongoing work and research directions

The work described has shown that several performance issues need to be addressed in the deployment of a Grid resource monitoring and discovery service. Firstly, the behaviour of the information gathering methods at the resource level (GRIS) is studied and conclusions deduced. Secondly, the performance which the client observes from an aggregate information server (GIIS) is assessed using the same metrics. These two sets of results are then compared to analyse the effect on the observable performance of querying an aggregate information server rather than an information server.

Based on the performance data collected when an aggregate information server is queried, we are also predicting performance metrics values using different algorithms. A client transparently querying an information server is desirable. Subsequently, it should not impact on the behaviour of the client in any way. This impact can be minimally reduced by knowing how long it will take to get back the required information. Performance prediction information is also necessary when users are faced with the issue of choosing to query one information server over another, without knowing how they will respond.

We are also extending the performance prediction framework to include PACE (performance analysis and characterisation environment) [19] application and hardware models, which can further be used in helping trust agents [20] select resources for job execution. Moreover, we will be focusing on the emerging OGSA (open grid services architecture)-based [21] MDS3 (monitoring and discovery system) [22] and the way in which the findings gathered can be applied to this new framework. The overall aim of this work is to ascertain the quality-of-service of Grid middleware to help deliver computational power to geographically distributed locations.

9 Summary

The experiments in this paper have shown the effect on the performance of querying the GRIS, of various information provider gathering methods and GRIS back-end implementations. Tuning the GRIS by adopting the most efficient evaluation method can greatly improve the performance obtained. Furthermore, carefully choosing the implementation language for the information providers and the type of relational database can enhance performance. Every GRIS back-end implementation needs to take into account factors including the number of simultaneous queries, information freshness and load on the GRIS node. The overriding importance of any one factor will influence the type of evaluation method to be used.

The information service behaviour observed when queries are sent to a GIIS has been analysed and compared with that of a GRIS. Several scenarios have been set up with different GRIS back-end implementations, and the experiment results demonstrate that caching is required at the higher levels of the MDS hierarchy for an acceptable level of performance to be obtained. A better performance is also attained when a GIIS is queried, rather than a GRIS; this happens when caching is enabled in the GIIS and the TTL value is at least 60 s. Furthermore, the value of the cache TTL depends on the expected number of users concurrently querying the GIIS.

10 Acknowledgments

This work is sponsored in part by funding from the EPSRC e-Science Core Programme (contract no. GR/S03058/01), the NASA AMES Research Centres (administered by USARDSG, contract no. N68171-01-C-9012) and the EPSRC (contract no. GR/R47424/01).

11 References

1 Foster, I., and Kesselman, C.: 'The GRID: blueprint for a new computing infrastructure' (Morgan Kaufmann Publishers, Inc., San Francisco, 1999)

2 Foster, I., Kesselman, C., and Tuecke, S.: 'The anatomy of the grid: enabling scalable virtual organizations', *Int. J. Supercomput. Appl.*, 2001, **15**, (3), pp. 200–222

3 Berman, F., Fox, G.C., and Hey, A.J.G. (Eds.): 'Grid computing: making the global infrastructure a reality' (Wiley, Chichester, 2003), pp. 557

4 The Globus Project. http://www.globus.org, accessed 11 July 2003

5 Legion. http://legion.virginia.edu, accessed 11 July 2003

6 The Condor Project. http://www.cs.wisc.edu/condor/, accessed 11 July 2003

7 Ninf: A global computing infrastructure. http://ninf.apgrid.org/welcome.shtml, accessed 11 July 2003

8 Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman, C.: 'Grid information services for distributed resource sharing'. Proc. 10th IEEE Int. Symposium on High-performance distributed computing (HPDC-10), San Francisco, CA, 7–9 August 2001, pp. 181–194

9 Aloisio, G., Cafaro, M., Epicoco, I., and Fiore, S.: 'Analysis of the globus toolkit grid information service'. GridLab-10-D.1-0001-GIS_Analysis, GridLab Project. http://www.gridlab.org/Resources/Deliverables/D10.1.pdf, accessed 11 July 2003

10 Zhang, X., Freschl, J., and Schopf, J.M.: 'A performance study of monitoring and information services for distributed systems' Proc. 12th IEEE Int. Symp. on High-performance distributed computing (HPDC-12), Seattle, WA, 22–24 June 2003, IEEE Press, pp. 270–281

11 Lim Choi Keung, H.N., Wang, L., Spooner, D.P., Jarvis, S.A., Jie, W., and Nudd, G.R.: 'Grid resource management information services for scientific computing'. Proc. Int. Conf. on Scientific and engineering computation (IC-SEC), Singapore, 3–5 December 2002

12 Globus: Extending GRIS functionality. http://www.globus.org/mds/extending-gris.html, accessed 11 July 2003

13 Fitzgerald, S., Foster, I., Kesselman, C., Laszewski, G., Smith, W., and Tuecke, S.: 'A directory service for configuring high-performance distributed computations'. Proc. 6th IEEE Symp. on High-performance distributed computing (HPDC-6), Portland, OR, 5–8 August 1997, pp. 365–375

14 UK eScience Programme. http://www.research-councils.ac.uk/escience/, accessed 11 July 2003

15 NASA's Information Power Grid (IPG). http://www.ipg.nasa.gov/, accessed 11 July 2003

16 von Laszewski, G., Foster, I., Gawor, J., and Lane, P.: 'A Java commodity grid kit', *Concurrency Comput. Pract. Exp.*, 2001, **13**, (8–9), pp. 643–662

17 Howes, T., and Smith, M.: 'LDAP: programming directory-enabled applications with lightweight directory access protocol', Macmillan Technology Series (Macmillan Technical Publishing, Indianapolis, 1997)

18 Lee, R., and Seligman, S.: 'JNDI API tutorial and reference: building directory enabled Java™ applications' (Addison-Wesley, Boston, 2000)

19 Nudd, G.R., Kerbyson, D.J., Papaefstathiou, E., Perry, S.C., Harper, J.S., and Wilcox, D.V.: 'PACE: a toolset for the performance prediction of parallel and distributed systems', *Int. J. High Perform. Comput. Appl.*, 2000, **14**, (3), pp. 228–251

20 Dyson, J.R.D.: 'Trust and negotiation management for grid computing'. Research Report, Department of Computer Science (University of Warwick, 2003)

21 Foster, I., Kesselman, C., Nick, J., and Tuecke, S.: 'The physiology of the grid: an open grid services architecture for distributed systems integration'. Open Grid Services Infrastructure (OGSI) Working Group, Global Grid Forum 5, Edinburgh, Scotland, 21–24 July 2002, http://www.globus.org/research/papers/agsa.pdf, accessed 11 July 2003

22 Information services in the globus toolkit 3.0 release. http://www.globus.org/mds/, accessed 11 July 2003