Predicting the Performance of Globus Monitoring and Discovery Service (MDS-2) Queries

Hélène N. Lim Choi Keung, Justin R.D. Dyson, Stephen A. Jarvis, Graham R. Nudd
High Performance Systems Group
Department of Computer Science, University of Warwick
Coventry, UK
hlck@dcs.warwick.ac.uk

Abstract

Resource discovery and monitoring in a distributed Grid environment gives rise to several issues, one of which is the provision of reliable performance and hence, the quality-of-service delivered to Grid users. This performance requirement brings about the necessity to know how the Monitoring and Discovery Service (MDS) would respond to various queries. This paper focuses on the performance of the MDS as part of the knowledge needed by a Grid entity, to choose a Grid Index Information Service (GIIS) for discovering resources. Several performance metrics are defined and the performance achieved by a GIIS is evaluated against that obtained by a GRIS [15]. Based on the GIIS performance data collected, the values for the performance metrics are predicted using different algorithms. Thus, past performance observations can be used to qualitatively characterise the future performance of the GIIS, allowing Grid middleware built on these services to be more predictable.

1. Introduction

Grid computing [12] enables collaborations amongst scientists in the form of data sharing and remote processing [13]. End users expect a reliable quality-of-service from Grid middleware and from continually improving network bandwidths. End-to-end quality-of-service depends on the reliable performance [6] obtained from each component of Grid middleware; an integral part of these being the Grid Information Service. This is an important component of Grid middleware since it enables the discovery and management of resource entities on the Grid. The contribution made by this paper is the characterisation of the performance expected from the Globus [3] Monitoring and Discovery Service (MDS) [9] which is a widely deployed reference implementation of a Grid information service.

Making use of a Grid Information Service like the MDS raises questions about the efficiency with which the service can be accessed. The answer depends on many factors, including the physical features of the machine on which the MDS is running, the load on the machine, and the characteristics of the resources on which information providers are running. This paper investigates the behaviour of the GIIS in responding to a query via several typical scenarios, and will focus on its end-to-end efficiency from the point where the query reaches the MDS, to when the client receives the response. Consequently, whilst network monitoring is indeed part of this resource discovery process, this paper analyses whether the behaviour of the MDS itself can be characterised and predicted. Therefore, a LAN is used to minimise external conditions affecting the performance of the MDS. Performance prediction of the network component of Grid computing has already been carried out [19].

The performance obtained from a GIIS depends on the performance of the GRISes which are registered with it. The performance is also dependent on the time-to-live (TTL) [11] of the information the GRISes return to the GIIS. Usually, a quick response can be expected when the data is cached. However, when the requested data has 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 the performance obtained with various information providers and GRIS back-end implementations has been reported in a previous paper [15].

Since the observable performance of a query to a GIIS relies on the complexity of the GIIS hierarchy, it is crucial to analyse and understand the performance of a simple GIIS hierarchy. This paper investigates the observable performance obtained when a user-level application queries a GIIS which has a single GRIS registered to it. Both the GIIS and GRIS are on the same server machine. Furthermore, the GRIS has the default core information providers supplying

information to it [15]. As a consequence, specific GIIS and GRIS implementations can be recommended, based on the performance prediction for MDS2 as derived from past observed performance data. Several different predictors are discussed and the way they are applied to previous performance data is analysed.

The rest of this paper is organised as follows; related work on which this paper builds, is presented in Section 2. Section 3 explains the experimental set up for gathering GIIS performance data. The experiments and their results are shown in Section 4, and Section 5 focuses on the various performance evaluation and prediction techniques applied to the MDS. The paper concludes with future work in Section 6.

2. Related work

The work in this paper builds on [15, 16] where the performance achievable from querying a GRIS with different back-end implementations, was analysed. A number of performance metrics were defined for the characterisation and comparison of performance. Results were obtained and conclusions drawn from them; here, the performance of a GIIS is evaluated, where the GIIS is located on the same machine as the GRIS. Results will show the difference in querying the GIIS using a number of evaluation methods.

Grid Information Service (GIS) performance had been previously examined. For example, security-enabled queries were sent to the MDS and compared to queries with no security involved [5]. Moreover, the performance of the MDS with different versions of LDAP [14] has also been investigated [17]. The results from [17] have contributed to the more recent MDS architecture. Furthermore, Schopf [20] examined the difference in scalability obtained from three information and monitoring systems: MDS2.1, R-GMA and Condor's Hawkeye system; scalability results are obtained when various system components are subjected to increasing user loads.

3. Experimental Setup

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 utilised in the majority of UK e-Science projects [4] and US testbeds including NASA's Information Power Grid [2]. Across the various experiments, the following agent setup was maintained where the agents were written using the Java CoG Kit libraries [18]. 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 ten minutes, 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. The *mscs* machines each has the Linux operating system installed 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 are connected to the GIIS host by a 100 Mb link. The time taken for each request to be serviced is measured and an average response time is calculated. Moreover, every agent sleeps for one second before sending the next request.

To test the scalability of the MDS, a GIIS and a GRIS were both set up on a Linux kernel 2.4.18-14 machine (M_1) which has a 1.9 GHz processor and 512 MB RAM.

In these sets of experiments, complex searches on data objects are not being performed. Subsequently, queries should return all the data available, thus allowing the status of the Grid to be discovered.

3.1. Evaluation Methods

Different implementations for the GRIS back-end and information providers were used in the experiments. Three definitions are given to the different GRIS evaluation methods [1]:

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 freshly generated information (lazy evaluation) and cache the information.

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), but it is readily available and is frequently updated.

3.2. GRIS Back-end Implementations

Using the evaluation methods discussed above, a number of implementations were set up for the GRIS back-end and a series of experiments carried out. These back-end implementations are:

- **Lazy evaluation (LE)**
  The GRIS cache TTL is equal to zero. On the 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. Moreover, the cache is filled with the new generated data.

- **Java speculative evaluation (SE)** The GRIS cache TTL is equal to zero and the information providers write their data to a relational 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 and two databases are used for this evaluation method, PostgreSQL and MySQL. The version of PostgreSQL used is 7.2.3 with no extra features such as caching enabled. Moreover, the version of MySQL used is 4.0.1 and caching is disabled.

- **Eager & Java speculative evaluations**
  The GRIS cache TTL is not equal to zero and the information providers write their data to a relational database on average every minute. The information provider accessors are written in Java and the same two relational databases are used.

- **Perl speculative evaluation**
  The GRIS cache TTL is equal to zero and the information providers write their data to a relational database 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. The same two relational databases are used.

When speculative evaluation is used as the GRIS backend implementation, the database does not contain any other records apart from the output from the core information providers.

3.3. Performance metrics

The same performance metrics as in [15] are used for consistency and comparison. They 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-minute load average ($\mathcal{L}_1$);
- 5-minute load average ($\mathcal{L}_5$).

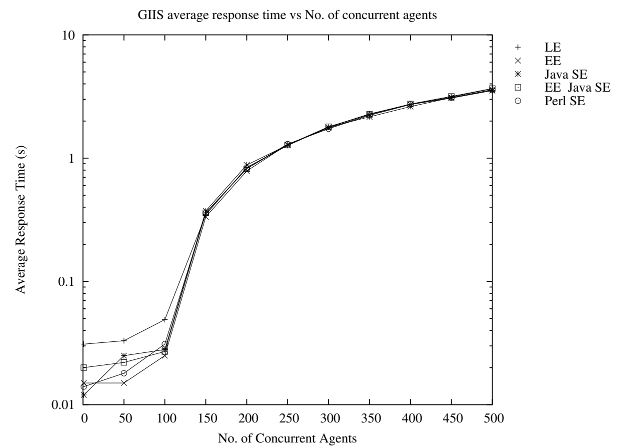

Figure 1. Experiment average response time.

4. Experiment Results

4.1. GIIS Results

This experiment examines the change in the scalability of a GIIS with an increase in the number of agents.

The timeout on the GIIS has been set to 600 seconds to allow time for any query response to return to the agent. Also, the GIIS cache TTL has been set to 3600 seconds to simulate the overhead of querying a GIIS as compared to a GRIS.

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 60s, the GIIS performance declines with more than 100 concurrent agents. Increasing the GIIS cache TTL further to 1800s, allows up to 200 agents to concurrently query the GIIS with adequate performance. Therefore, to obtain an acceptable level of performance from the GIIS and to stretch its resource discovery behaviour, its cache TTL has been set to 3600s throughout this experiment. This cache TTL value is reasonable because the nature of the resource information provided by the core information providers is rather static, for instance the type of operating system, the total memory available and the CPU model.

Up to 500 agents queried the GIIS simultaneously, with a waiting period of one second between receiving a query response and issuing the next query. There were a maximum of 50 concurrent agents on each of the ten *mscs* machines, issuing queries over a period of ten minutes. The results from the experiment are shown below.

Figure 1 shows that all five implementation methods allowed relatively consistent average response time to be obtained when 1 to 150 agents simultaneously query the GIIS. For clarity, only the results for PostgreSQL are shown, as

the results with MySQL are similar. Lazy evaluation has the highest $\mathcal{R}_T$ with an average of 0.03s; the four other GRIS back-end implementation methods have much lower $\mathcal{R}_T$. EE displays the least $\mathcal{R}_T$ and the relatively sharp increase in $\mathcal{R}_T$ for Java SE and, for EE and Java SE, can be attributed to database communication and the overhead of the JVM. When the number of concurrent agents increase beyond 100, the $\mathcal{R}_T$ of all the methods are comparable and increase quadratically up to about 3.7s. This effect is the result of caching in the GIIS; the extra overhead on the GIIS when there are more agents, is handled uniformly by the GIIS cache as the data is almost always in cache.

The corresponding throughput graphs, which are not shown, are the inverse of those in Figure 1 and they indicate that caching data or updating it periodically in a database at the GRIS backend, enables the GIIS to handle more queries per second. The throughput levels at 3.0 queries/s when 150 agents concurrently query the GIIS; it further drops to around 0.3 queries/s for 500 agents.

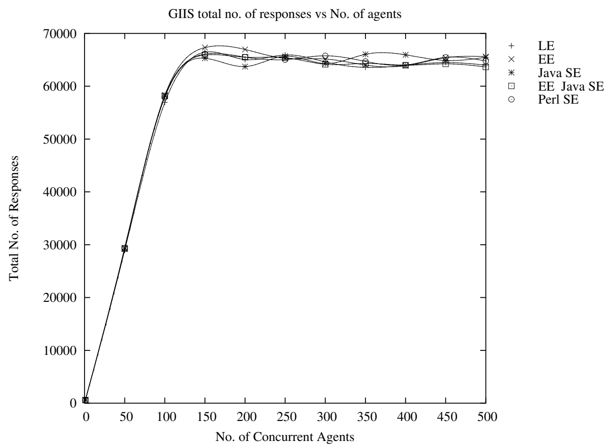

Figure 2. Experiment total number of responses.

Figure 2 shows that all evaluation methods return very similar numbers of query responses for the various numbers of agents. The $\mathcal{R}_e$ for 1 to 10 agents increases slowly to just over 5000 and it increases sharply to around 65000 at 150 agents. As the number of agents keeps increasing, the $\mathcal{R}_e$ stays stable at 65000. If less than 150 agents simultaneously query the GIIS, $\mathcal{R}_e$ is increasing because more queries can be answered per second. Nevertheless, as more agents hit the GIIS, the latter reaches its saturation point and cannot process more than 65000 requests over an average of 10 minutes.

In Figure 3, the 1-minute load average of Perl SE, EE and LE start off at about 0.2 and they almost quadratically increase to 8.0 for 100 agents. From that point, Perl SE $\mathcal{L}_1$ increases to about 40.0 while EE and LE $\mathcal{L}_1$ stabilise at

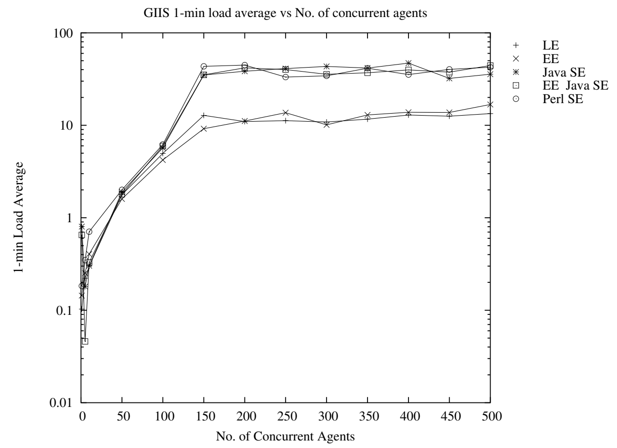

Figure 3. Experiment 1-min load average.

around 11.0. This is because of the communication which Perl SE has with the PostgreSQL database. On the other hand, Java SE and, EE and Java SE $\mathcal{L}_1$ decrease slightly as the number of agents increases from 1 to 5. This is due partly to the GRIS caching effect and to the low number of queries received. But with an increase in the number of agents, their $\mathcal{L}_1$ increases from about 0.3 to 40.0. This increase stops with the number of agents rising from 150 to 500. The overhead from the Java language and the periodic writing to the PostgreSQL database raises the load to a relatively higher number. The curves on the 5-min load average graph are not shown because they follow the same trend, with the exception that the minimum load average is 0.1. On average, $\mathcal{L}_5$ will be greater than $\mathcal{L}_1$.

4.2. Comparison of GIIS and GRIS Performance

Figures 4 to 7 show the differences in the performance of a GRIS and GIIS with increasing concurrent requests. Figure 4 shows that 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 3.5s 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), indicat-

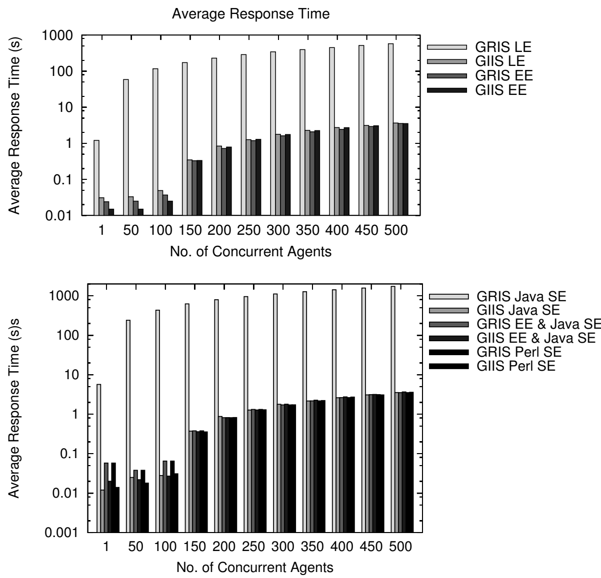

Figure 4. Comparison of GIIS and GRIS average response times.

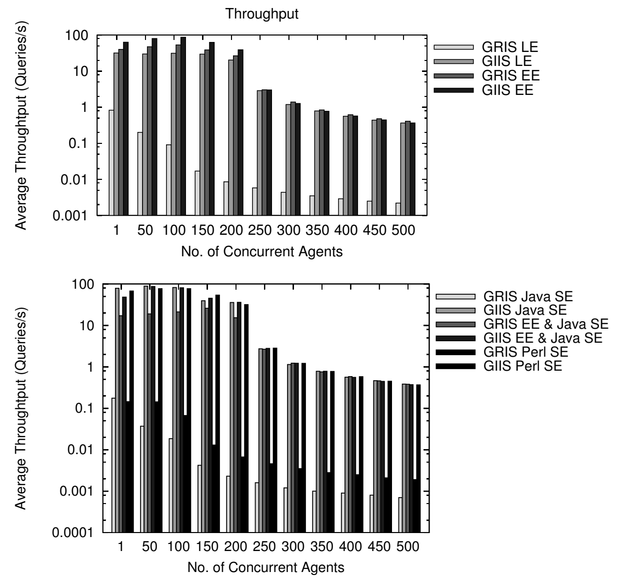

Figure 5. Comparison of GIIS and GRIS throughputs.

ing that caching in the GIIS levels out any differences in $\mathcal{R}_T$ from the GRIS. Figure 5 is related to Figure 4 and is shown for reference.

The total number of responses with GIIS LE sharply increases with the number of agents, and starts to stabilise for a number of agents greater than 100 at around 66000. Moreover, querying the GIIS instead of the GRIS allows more responses to be serviced. The GRIS and GIIS values of $\mathcal{R}_e$ for EE are very similar, indicating that caching in both the GRIS and GIIS can allow for a maximum number of simultaneous queries to be serviced. Similar results are found for both Java SE and Perl SE as for LE. The results for EE and Java SE (PostgreSQL) are similar to those with EE. All these results also show a constant $\mathcal{R}_e$ across the different experiments, suggesting that there is a maximum number of queries which the GIIS can handle under its set conditions.

The GIIS 1-minute 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 on the node 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. Results which are not shown here, indicate a similar behaviour for LE $\mathcal{L}_5$. With less than 50 agents, caching in the GIIS allows the load on the node to be low 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 produces higher $\mathcal{L}_1$ and $\mathcal{L}_5$ due to the overhead incurred by SE.

5. Performance Evaluation and Prediction

5.1. Predictive Methods

It is difficult to observe a trend in the behaviour of the GIIS; therefore, a number of predictive methods are applied to past GIIS data. In addition, the characterisation of the performance of the MDS does not take into account the network load or the Grid topology. The approach taken is to formulate a prediction for the GIIS performance, for the benefit of a Grid application. A performance methodology is therefore developed to choose the most appropriate GRIS evaluation method and for predicting the cost of queries. The performance prediction of a query from observations of past queries can be used by the end user, that is the agent, which is interested in the resource discovery end-to-end performance. This performance information can be used in its other functionalities, including metascheduling and contributing to the guarantee of quality-of-service contracts.

The performance prediction of a query to the MDS is based on collecting performance information for each query that is made to the GIIS and then applying predictive methods to the previous observations. The gathering of per-

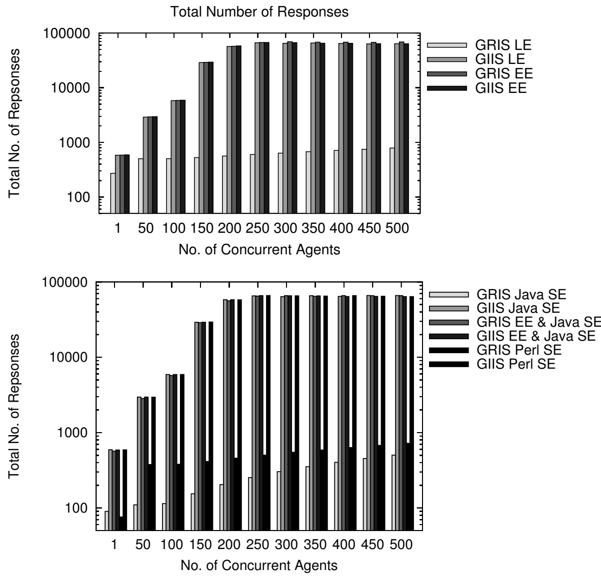

Figure 6. Comparison of GIIS and GRIS total number of responses.

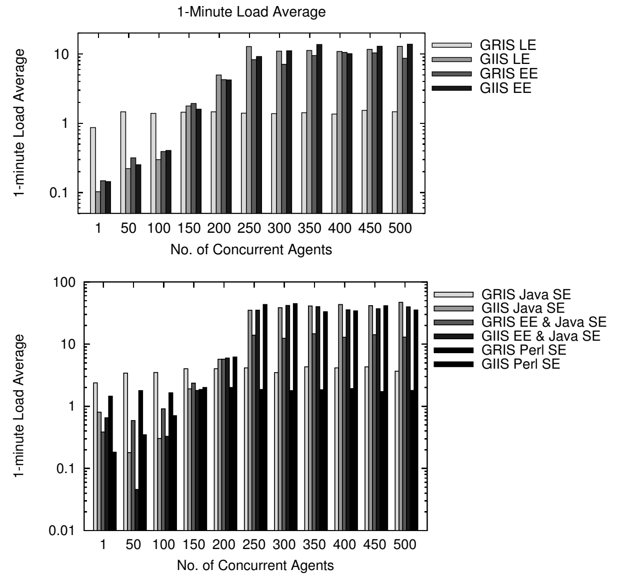

Figure 7. Comparison of GIIS and GRIS 1-minute load average.

formance information does not affect the behaviour of the MDS in any way. The different predictors [10] used to estimate future query times are:

- **Last observation** The most recent, single performance observation value is taken as the prediction. The last performance value is most likely to reflect the behaviour of future queries.

$$P_n = V \tag{1}$$

- **Sample average** The prediction is the mean average of the past performance values within a sample set. This set is defined by a sliding window of size *x*, which corresponds to the x most recent observations. Not all the performance values are used for the average as old values become less relevant. This predictor is used when performance information is produced on a regular basis. However, given a fixed performance data set, an average can be used with a maximum window size.

$$P_n = \frac{\sum_{i=1}^{x} V_i}{x} \tag{2}$$

- **Low pass filter** Recent performance data constitutes a better predictor than older data. Subsequently, this predictor uses an exponentially degrading function to obtain an average of the recent performance behaviour of the MDS. This is achieved by using the *low pass filter* formula:

$$P_n = (w \bullet P_{n-1}) + ((1-w) \bullet V) \tag{3}$$

where

P_n is the prediction and the new value of the low pass filter
P_{n-1} is the previous filter value
V is the most recent performance observation value
w is the weighting parameter and is a value between 0 and 1
x is the size of the sliding window

The value of w is 0.95 [10], thus decreasing the value of the weight as observation values grow older and increasing the prediction accuracy.

- **ARIMA models** Due to the fact that the observed data is *stationary* i.e it varies about a fixed level, *ARIMA* (autoregressive integrated moving average) [7, 8] models are used to project the data to produce forecasts. The two most adequate models have been identified and are the AR(1) and AR(2) models. In the AR(1) model, forecasts for the next value depend on the observations in the previous time period; whilst in AR(2) models, forecasts of the next value depend on observations in the two previous time periods.

5.2. Query Performance Prediction Results

Figures 8 and 9 show the results obtained when the average time for a query is predicted using the different pre-

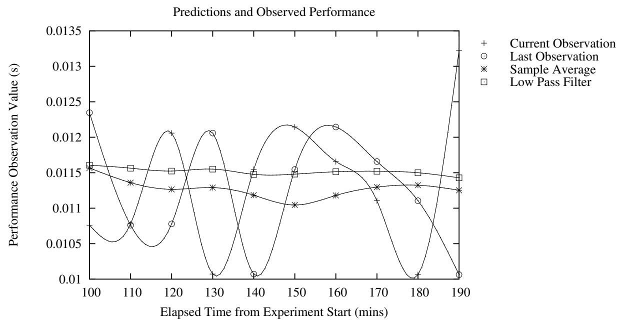

Figure 8. Actual observed data and predictions.

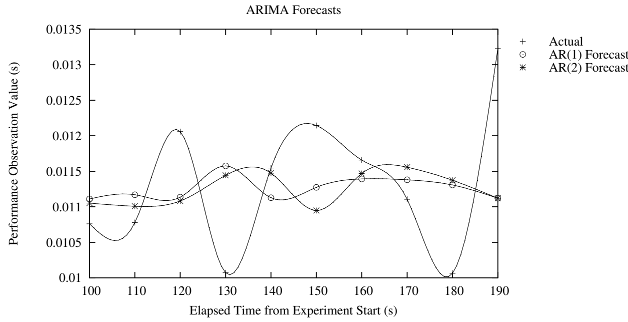

Figure 9. Actual observed data and ARIMA forecasts.

dictive methods. In addition, the actual performance observation data is also shown. In the experiment, one agent repeatedly queries the GIIS using the EE implementation, and the average response times collected. A sliding window of ten performance observation values are used for the *sample average* predictor. Moreover, the mean of the sample data set is used as the initial prediction for the *low pass filter* predictive method. Query performance prediction is started when 100 minutes have elapsed since the beginning of the experiment.

The one-step ahead AR(1) and AR(2) forecasts which are made over an increasing time series, are shown in Figure 9. These two models have been checked for residuals and are considered adequate for forecasting. The graphs show that the two kinds of predictions closely fit the actual observed time series.

The predicted and actual values for $\mathcal{R}_T$ at each recorded time, which are shown in Figures 8 and 9, indicate various levels of prediction accuracy. For instance, after 120 mins have passed since the start of the experiment, the observed data is 0.012s, *last observation* predicts 0.010s, and *sample average*, *low pass filter* and both *ARIMA* methods predict a time of 0.011s. Furthermore, since the observed data tends

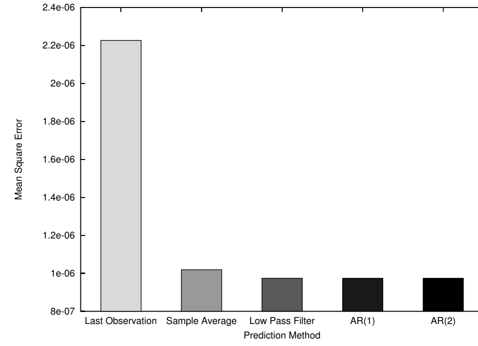

Figure 10. Mean square error of predictions.

to vary greatly from one value to the next, the *last observation* prediction which follows it, is rather inaccurate. On the other hand, the *sample average* and *low pass filter* predict values which fluctuate rarely.

To compare the prediction efficiency of the different predictors, their mean square error (MSE) is calculated, using the equation below:

$$MSE = \frac{1}{N}\sum_i (O_i - P_i)^2 \tag{4}$$

where

N is the data sample set size
O is an observation value
P is a prediction

Figure 10 shows the resulting MSE values of the predictions with respect to the observed performance values. The bar chart shows that *last observation* has the highest MSE, and that *low pass filter* has the lowest. These results are expected because the *last observation* does not necessarily guarantee that the actual observed value would be the same. The prediction is improved by 54% with the *sample average* predictor which takes into account previous performance observations. There is a further 4% improvement in prediction with the *low pass filter* as it considers only the average recent behaviour of the MDS. The *low pass filter* equation exponentially decays the significance of older data and it is slightly more accurate than the *ARIMA* models.

6. Conclusions and Future Work

The approach taken by this paper is to predict the behaviour of the MDS from a Grid application's point of view, based on past performance data. These predictions are used to help the application decide which GIIS to choose for sending queries. This informed choice further contributes to

guarantee quality-of-service in the use of Grid middleware. To do so, the performance achievable 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 information providers and GRIS back-end implementations. The experiment results demonstrate that caching i.e EE, is required at the higher levels of the MDS hierarchy for an acceptable level of performance to be obtained. It has also been found that GIIS caching can annul the benefit of using GRIS caching. Furthermore, a better performance is obtained when a GIIS is queried, rather than a GRIS when caching is enabled in the GIIS and is at least 60s. The value of the cache TTL depends on the expected number of users concurrently querying the GIIS.

Using past MDS performance observation data, several predictive algorithms are implemented and the experiment results analysed. It has been found that the accuracy of the predictors were clearly different. The *low pass filter* predictive algorithm was the most accurate method, with the *sample average* predictor being only slightly less accurate. Even though the *last observation* method was the least accurate, its MSE was of the order of $2\text{x}10^{-7}$, which is very low. More sophisticated forecasting models have also been applied to the observed data using AR(1) and AR(2).

Future work will include applying the predictive mechanisms of the MDS to other higher-level Grid middleware components, for example Grid agents. The aim will be to ascertain quality-of-service characteristics of Grid middleware in its promise to deliver computational power to geographically distributed locations.

7. Acknowledgements

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).

References

[1] Globus: Extending GRIS Functionality. http://www.globus.org/mds/extending-gris.html.

[2] NASA's Information Power Grid (IPG). http://www.ipg.nasa.gov/.

[3] The Globus Project. http://www.globus.org.

[4] UK eScience Programme. http://www.research-councils.ac.uk/escience/.

[5] G. Aloisio, M. Cafaro, I. Epicoco, and S. Fiore. 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.

[6] F. Berman, G. C. Fox, and A. J. G. Hey, editors. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003. page 557.

[7] G. E. P. Box and G. M. Jenkins. *Time Series Analysis Forecasting and Control*. San Francisco: Holden-Day, 1st edition, 1970.

[8] G. E. P. Box, G. M. Jenkins, and G. C. Reinsell. *Time Series Analysis: Forecasting and Control*. Englewood Cliffs, N.J.: Prentice-Hall, 3rd edition, 1994.

[9] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. *In Proc. 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, pages 181–194, 7-9 August 2001. IEEE Press.

[10] N. Dushay, J. C. French, and C. Lagoze. Predicting indexer performance in a distributed digital library. *Third European Conference on Research and Advanced Technology for Digital Libraries (ECDL'99), Paris, France)*, September 1999.

[11] S. Fitzgerald, I. Foster, C. Kesselman, G. Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. *In Proc. 6th IEEE International Symposium on High-Performance Distributed Computing (HPDC-6)*, pages 365–375, 5-8 August 1997. IEEE Press.

[12] I. Foster and C. Kesselman. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.

[13] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. Supercomput. Appl.*, 15(3):200–222, 2001.

[14] T. Howes and M. Smith. *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*. Macmillan Technical Publishing, 1997.

[15] H. N. Lim Choi Keung, J. R. D. Dyson, S. A. Jarvis, and G. R. Nudd. Performance evaluation of a grid resource monitoring and discovery service. *IEE Proc.-Software*, 150(4), August 2003.

[16] H. N. Lim Choi Keung, L. Wang, D. P. Spooner, S. A. Jarvis, W. Jie, and G. R. Nudd. Grid resource management information services for scientific computing. *International Conference on Scientific & Engineering Computation (IC-SEC) 2002, Singapore*, 3-5 December 2002.

[17] W. Smith, A. Waheel, D. Meyers, and J. Yan. An evaluation of alternative designs for a grid information service. *In Proc. 9th IEEE International Symposium on High-Performance Distributed Computing (HPDC-9)*, pages 185–192, 1-4 August 2000. IEEE Press.

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

[19] R. Wolski. Dynamically forecasting network performance using the network weather service. *Journal of Cluster Computing*, 1:119–132, January 1998.

[20] X. Zhang, J. Freschl, and J. M. Schopf. A performance study of monitoring and information services for distributed systems. *In Proc. 12th IEEE International Symposium on High-Performance Distributed Computing (HPDC-12)*, pages 270–281, 22-24 June 2003. IEEE Press.