Dynamic Discovery of Available Resources in A Computational GRID Environment

Kevin Curran, Gerard Parr and Alan Bradley

School of Computing and Intelligent Systems
Faculty of Engineering, University of Ulster, Northern Ireland
Email: kj.curran@ulster.ac.uk

ABSTRACT

Corporations such as Boeing are currently using computational GRIDs to improve their operations. Future GRIDs will allow an organisation to take advantage of computational GRIDs without having to develop a custom in-house solution. GRID Resource Providers (GRP) make resources available on the GRID so that others may subscribe and use these resources. GRPs will allow companies to make use of a range of resources such as processing power or mass storage. However simply providing resources is not enough to ensure the success of a computational GRID. Access to these resources must be controlled otherwise computational GRIDs will simply evolve to become a victim of their own success, unable to offer a suitable Quality of Service (QoS) to any user. The task of providing a standard querying mechanism for Computational GRID Environments (CGE) has already witnessed considerable work from groups such as the Globus project who have delivered the Metacomputing Directory Service (MDS) which provides a means to query devices attached to the GRID. We present a monitoring component both capable of the dynamic discovery of available resources in a computing GRID environment and executing user jobs on the available resource at a given moment in time. We show that it is possible to construct a monitor based on the CoG toolkit and MDS to monitor the available resources attached to a CGE.

Keywords: Computational GRID, Grid Resource Providers

1 INTRODUCTION

Computational GRIDs have emerged as the next phase in distributed computing. They offer a degree of resource sharing that will surpass even the World Wide Web (WWW) as they will not only change the way in which data is accessed but also how this data is produced, consumed and stored. GRID computing aims to make all computing resources available constantly on a 24/7 basis [Foster00]. Many large corporations such as Boeing are currently using computational GRIDs to improve their operations. However future GRIDs will allow an organisation to take advantage of computational GRIDs without having to develop a custom in-house solution. GRPs will allow companies to make use of a range of resources such as processing power or mass storage. However simply providing resources is not enough to ensure the success of a computational GRID. Access to these resources must be controlled otherwise computational GRIDs will simply evolve to become a victim of their own success, unable to offer a suitable Quality of Service (QoS) to any user. If too many users utilise resources without contributing, then excess burden is placed on a system and eventually if enough users abuse the system in this manner the overall performance for everyone will degrade [Hardin68, Oram01]. The same situation could easily arise with computational GRIDs if users were simply to consume resources without contributing any in return. Some proposals for a 'points' based scheme in which users earn points by providing resources and can hence consume resources based upon the 'points' they have accumulated. This scheme would be adequate if all points and users on the GRID were equal and thus capable of providing the same volume of resources that they need to consume. As every GRP will have various pricing mechanisms and each will offer slightly different resources, an organisation will therefore require an effective means of discovering available GRPs and constantly monitoring performance so that they can dynamically change the current GRP when performance falls below acceptable limits. This research seeks to develop a system which allows users to discover available resources on a GRID and then use these resources while monitoring the performance of the resources over time to ensure that the required quality of service (QoS) is delivered. In the event of the QoS falling below acceptable limits this system will also be capable of discovering similar services on the GRID that can meet the desired QoS levels therefore allowing user applications to dynamically discover the best possible GRP at the time of execution.

The purpose of this research is to lead to the construction of GRID environments in which user application can discover and monitor the performance levels offered by various GRPs attached to the GRID. Upon discovering better performance levels at an alternative GPR the user's application would dynamically reconfigure itself to run using the resources which offer better performance levels. For a system like this to be effective it will require a front-end that allows the users to define a set of performance parameters that can be used to check the performance of each GRP. These parameters may not be a simple fact of hardware performance levels but will encompass information relating to pricing so although better performance maybe available at another GRP it maybe out of the price limits of the current user and therefore the application would not make use of those

resources. This work focused on the task of discovering available GRID services and continually monitoring the performance levels they provide. As GRID awareness grows so to will the demand for GRID applications and services however not all organisations will be able to afford the construction of their own global GRIDs. Therefore this research seeks to develop a solution that will allow an organisation to discover what resources are publicly available on the GRID, it will also ensure a consistent QoS can be obtained from these services by monitoring their performance levels and when necessary recommending an alternative service offering better performance.

2 GRID SOFTWARE

No matter how powerful the infrastructure behind a computational GRID is it will be nothing without software that allows users to take full advantage of the technology. Typically batch jobs which require a significant amount of processing will be best suited to take advantage of the GRIDs resources. Development of applications capable of running on the GRID will be required to run in a heterogeneous environment. Previous experience of distributed applications development has shown that Application Programming Interfaces (API), frameworks and middleware have facilitated the rapid development of distributed applications as they provide a level of abstraction that allow developers to concentrate on the application logic without concerning themselves how the application will use the underlying network. The successful and rapid development of GRID applications will require that the developer is afforded the same level of abstraction. The Globus project [Globus03] is an effort to build a set of essential GRID services for the construction of computational GRIDs. These services cover key aspects such as security, resource location, resource management and communication. The toolkits provision of these key services greatly helps in the development of GRID applications as developers can make use of existing services and are thus free to concentrate on the implementation of application specific logic [Foster00]. The Globus framework is based on a layered architecture in which high-level services are built on top of an essential set of core local services. The Globus Resource Allocation Management (GRAM) provides the local component for resource management [Czajkowski98]. GRAM provides a consistent API which can be used by GRID tools and applications to exchange requests for the allocation of resources. The Globus toolkit uses the Nexus communication library [Foster96]. The Nexus libraries define a low-level API that is used to support a range of higher-level programming models such as remote procedure call (RPC) and remote I/O. The Globus toolkit has been designed to support a wide range of communication types that will be used within computational GRIDs. It is equipped with an API that affords developers with a high degree of control over the mappings between high-level communication requests and the underlying protocol requests. This is the Nexus API [Foster96]. The Globus Metacomputing Directory Service (MDS) maintains dynamically updated information on the underlying communications network, protocols, network bandwidth and latency. Higher-level applications and libraries can use this information to configure the communications system in the manner most suited to the current application. The Globus Metacomputing Directory Service (MDS) is a repository of information relating to the state of the components in the GRID. This information is updated dynamically and can therefore be used by applications to adapt their behaviour to changes in the GRID. MDS provides a set of tools and APIs for discovering, publishing and accessing information about the structure and state of a GRID.

With the emergence of the Computing GRID Environment (CGE) new services are becoming available which go beyond those on offer in the current Internet. Although these new services are advantageous they come with challenges for applications as they may not be compatible with existing commodity technologies used to develop distributed applications. To integrate GRID technologies with existing commodity technologies the Globus project has developed Commodity GRID (CoG) kits [COG03] which allow developers to develop applications using various technologies and integrate with emerging GRID technologies. Currently CoG kits have been developed for a number of programming languages including Java, Perl and Python, each of these CoG kits provide interfaces for the appropriate language to interface with a GCE. Past experience of developing distributed applications has lead to a number of technologies such as CORBA [CORBA02], RMI, JINI [Arnold99] and DCOM [Rogerson97] which facilitate the development of distributed applications by providing a framework that provides common services needed by all applications. As a result the development of distributed applications was simplified as developers were able to make use of frameworks that had already been tried and tested. Technologies such as those mentioned above have been classified as commodity computing [Laszewski00]. The distinction between this and GRID computing is commodity technologies tend to focus on issues of scalability, component composition, and desktop presentation, while GRID developers emphasize end-to-end performance, advanced network services, and support for unique resources such as supercomputers. The development of CoG kits has therefore been an effort to bridge the gap between these two aspects of computing and allow the wide variety of commodity applications to take advantage of GRID technologies and services through appropriate interfaces [Laszewski00].

3 DISCOVERY OF GRID SERVICES

Here we examine existing approaches to discovering GRID services, monitoring their performance levels, and

accounting. Currently there has been a number of proposed solutions and implementations to the problem of maintaining a consistent QoS in a GRID environment, however many of these proposals have yet to be fully implemented. In addition, the problem of discovering services attached to the GRID will have to be addressed. In a global GRID consisting of many GRP an organisation wishing to establish a global GRID based on the services of others will need a way of discovering the best possible services available. Likewise each GRP will then require a means to bill users for using these resources. The problem of discovering services attached to a network has always been a concern for network engineers, in the past solutions included lookup and naming services and more recently technologies such as Sun Microsystems JINI have been developed to allow devices the capability of dynamically discovering the available services attached to a network. The MDS is provided by the Information Services component framework of version 3.0 of the Globus toolkit [MDS03]. The purpose of this framework is to provide information which can facilitate GRID resource discovery, selection and optimization. MDS gathers this information using the GRID Index Information Service (GIIS) and GRID Resource Information Service (GRIS). GIIS offers a means to obtain information on the entire collection of devices attached to the GRID, whereas GRID runs on the individual devices and allows clients to directly query the devices.

Monitoring services on offer within a GRID environment is important for a number of reasons. By conducting an analysis of the performance the service could be tuned to improve the performance levels. Also more importantly within a GRID environment monitoring of performance levels allows for performance predictions and therefore allows administrators to ensure their site will be capable of meeting the current demand. Monitoring will also allow better decisions to be made by GRID schedulers responsible for deciding where a job should execute. The Globus project uses the GRID Resource Allocation Management (GRAM) protocol for the task of resource management. The problem of managing resources in a GRID environment, problems can stem from distributed users and resources, variable resources states, variable grouping and connectivity and the lack of a centralized scheduling policy [GRAM03]. If GRID computing is to be successful on a commercial level GRPs will need an effective means to track usage of resources by individuals or smaller organisations so that the users of the resources can be charged and thus allowing GRPs to remain profitable and continue offering resources to the GRID.

4 SERVICE RESOURCE FRAMEWORK

GRPs in the future are likely charge users for access to their resources. As a result we feel that two classes of services will emerge. The first class will make use of reservation protocols and charge users a premium for guaranteed access to resources at the required time. The second class service is the one we will be concerned with for this paper, it will not make use of reservation protocols and will therefore make no guarantees about the processing time of a user's job, it will simply relay on discovering the available resources in a computational GRID environment. Such a system will require the ability to communicate with the GRID and request information about the resources attached to the GRID. It will then require the ability to take the details regarding an available resource and uses this information to execute all or part of a job on that resource. Therefore the system will require a monitoring component that monitors the GRID to determine the available resource and also a scheduling component that will have the responsibility of executing the jobs on the available resource. The user credentials are represented using digital certificates. These offer a secure and reliable means by which a user or organisation can be identified and therefore offers a plausible method upon which to build an accounting and billing system.

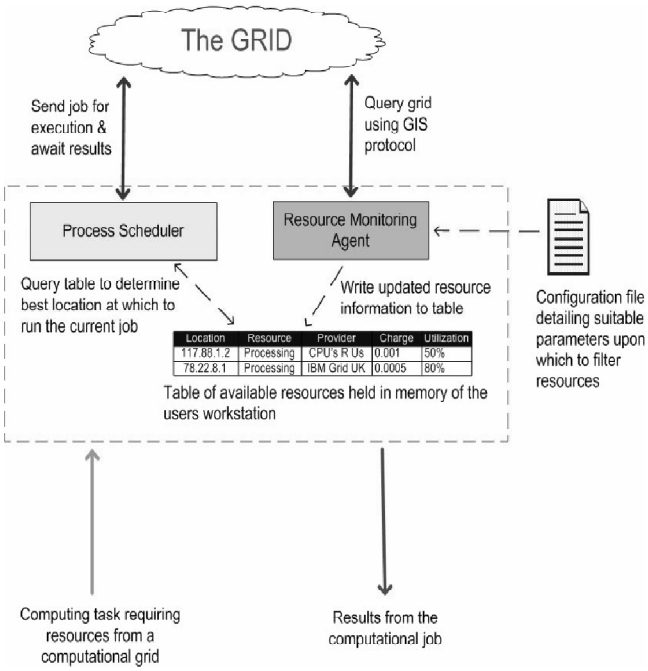

Figure 1: System Architecture

The design of this system adopts techniques from client/server computing. P2P techniques will be used to communicate with all centres on the GRID offering resources so that the availability of these resources can be determined. Once the client has found a centre that can execute its job it will initiate a process that will resemble client/server communications. It will send the job to be executed to the centre along with additional information such as user credentials. It will then wait for a response from the centre indicating if the job has completed or if

the job has been suspended to make way for a job with a higher priority. In the later case the system will then select an alternative centre to execute the job and wait for a response from this centre. This process continues until the job completes its execution.

4.1.1 Process Scheduler

In all but the simplest GRID implementation a scheduler is responsible for determining what job should run at each site. The scheduler reacts to the current availability of resources on the GRID and the priority of each job. The type of scheduling an application uses could offer a potential bases for the billing of GRID usage. For example a large organisation with substantial financial resources may be prepared to pay a premium rate for guaranteed access to the necessary resources when they need them. However a smaller organisation without the financial power may be happy to have their jobs executed on a scavenger system where there is no guarantee of access to the resources but cost is at a minimum. We have implemented a process scheduler which is responsible for controlling where the job is executed. This works in a similar manner to a thread scheduler used in most modern operating systems and will also have aspects in common with a scheduler which is used in GRID toolkits such as those provided by Globus [Gridbus03] and Sun Microsystems GRID Engine software [Sun03]. The unique feature of this scheduler will be in how it determines where the job should be run. As indicated in Figure 1 a table is held in memory, the scheduler will therefore query this table to find suitable resources that match those required by the current job. The process scheduler will also have to facilitate the process of identifying the user to the system so that resource usage can be billed to the appropriate user. In theory this will be achieved using digital certificates as this will facilitate integration with existing and proposed implementation for accounting software on the GRID.

4.1.2 Resource Monitoring Agent

The Resource Monitoring Agent is the key that is required to facilitate the dynamic discovery of resources and hence it is central to the development of a system that allows computational tasks to take advantage of the available resources without having to use resource reservation protocols. The agent is intended to use the GIS protocol included with the Globus toolkit. This is due to the fact the GIS service already provides a means to query resources attached to the GRID. Also the Globus toolkit is open source and therefore freely available to others to implement in their GRID toolkits. Also the software from Globus toolkit and Sun Microsystems has the ability to integrate with each other [Sun03]. Using the GIS protocol the agent will communicate with the GRID to retrieve the necessary information relating to each resource. This information will be filtered using parameters defined by the users in a configuration file. Examples of parameters could involve setting a lower and upper limit on the price of executing a computer cycle at a GRP. The results of this filtering will be stored in a table on the clients machine sorted the best available resource at the top of the table. This table will then be read by the 'Process Scheduler' so that it can execute all or part of the required task. If the table is empty it means the agent has been unable to find any suitable resources and therefore the 'Process Scheduler' will not be able to execute any tasks. The agent process will run at periodic time intervals updating the table each time it finds new resources or that existing resources have changed. The goal here is to produce a lightweight system that can be introduced to any platform so that users may take advantages of available resources within a CGE without the added expense of reserving the resource at a premium. Therefore the number of class and implementation will be kept to a minimum. Based on the decision to use the CoG kits provided by the Globus project due to their provision of a high-level Application Programming Interface (API) which allows application developers to build applications which make use of GRID protocols. As a result the choice of programming languages were limited and from this set the Java language was chosen due to the fact that it has already proved itself in the field of distributed systems. Also the cross platform capabilities offered by Java will allow the application to be developed so that it can be used by a wide variety of clients, an important fact in a heterogeneous environment such as a CGE. XML [XML03] has been chosen as the means to store the current set of available GRID resources. This is due to the fact all the necessary XML capabilities are included with the Java Runtime Environment (JRE). Also Java provides superb support for XML [Java03] in the form of Document Object MODEL (DOM) [Martin00] and Simple API for XML (SAX) [SAX03]. Since both the Monitor and the Scheduler will be using the XML file there is an issue of contention. In a system such as this it is felt that the updated information provided by the Monitor is more important and should therefore have priority when accessing the file. Hence this ensures the Scheduler will always be making use of the most up-to-date information. As a result the Singleton design pattern has been used to develop the ResourceTable. The *Singleton* design pattern is a design pattern that allows developers to construct classes which are restricted to having only one instance in the program. The pattern facilitates a global mechanism that can then be used by other classes to access this instance [Gamma03]. Here, the Singleton pattern has been used to construct all the classes as there should only be one instance of each class in the system. Also the Scheduler and Monitor will need to coordinate their effort by using the ResourceTable; this is achieved by the Singleton pattern which provides a global method that can be used to access the unique instance of the class. The overhead of a resource monitoring agent is balanced by the knowledge that as the global performance of computers increases, any overhead becomes a lower and lower percentage of the total resources used by the application [Khan03].

5 SYSTEM IMPLEMENTATION

The GRID monitoring class is a key component of the system. It is intended that this component will run continually updating the information of available resources on the GRID. This is implemented based on the Java Threads API which allows the process to run periodically. In order to interact with a computational GRID the monitor uses the Java CoG kit. In particular to fulfil the role of querying the GRID resources the classes in the org.globus.mds package were used; this provided a means to query both GRIS and GIIS services in a CGE. The experiments were conducted with Solaris 2.3 multi-threaded operating system installed on each experimental test bed node and each node interconnected by ATM switches and gigabit ethernet.

Once the monitor has retrieved the results from the MDS server they have to be made available to the scheduler so that it can determine the best resource to use in order to complete the current job. XML has been chosen as the means of holding the data due to its portability that means it can be passed easily between other processes for processing. In the case of this system it is planned that the retrieved data will be filtered further to ensure that it meets the limits imposed by users in a configuration file. However for now translating the data into XML will offer an effective means to store the retrieved results for later analysis. As previously stated the ResourceLookupTable class is a wrapper class used to abstract the details concerned with holding, accessing and using the results returned from the monitoring class. The ResourceTable class also makes use of a filter class to filter the information based upon parameters the user has set in a property file. As only one instance of the ResourceLookupTable is required by the program the class has used the Singleton design pattern to ensure this remains true throughout the execution of the program. As the Singleton pattern has been used all classes wishing to use the ResourceLookupTable will have to first call the getInstance() method, this checks if a new instance of the table has to be created, creates an instance if necessary and then returns a handle to the ResourceLookupTable to the calling class. As all interaction with the ResourceLookupTable involves first calling the getInstance method both the monitoring and scheduling class will be able to operate using the same resource table and therefore coordination between the two classes will be possible. As the resources will constantly change due to the monitor reporting updated details of the utilisation of resources in the GRID, the current set of resources will not be stored on disk, but will instead be stored in memory, thus reducing the overhead of disk I/O. Also there would be no point in storing these resources as they will only be valid during the current execution of the program, therefore any subsequent execution will have to discover the available resources again for themselves. As the data is being collected in XML the org.w3c.dom.Document object has been used to hold the details on current resources. This can then be filtered and sorted so that the first node could be passed to the scheduler that would then extract the data from the node and take the necessary steps to execute the current job using the given resource. The results must be filtered to ensure that they meet the needs of the current job being executed. Filtering is carried out by a class which implements the ResourceFilter interface; this interface defines the filter method. The reason why an interface was used is that interfaces allow many different implementations to be made each of which can be readily exchanged at runtime since they all share a common interface, and it is this common interface that a program interacts with. Therefore in a program such as this, interfaces would allow users to implement their own filters whenever appropriate. For the purpose of this implementation the implementation of the filter uses the Document Object Model (DOM) facilities included with JDK 1.4 to remove nodes from the XML document that do not meet the criteria of the current job. Although SAX gives better performance than DOM, DOM has been chosen here due to its higher degree of flexibility. With this implementation of filter users are able to specify parameters in a property file which are read by the filter at runtime and used to filter the XML. The property file could then be loaded into the program and accessed using the Properties class included in the java.util package. This class provides an effective means to access the values of a property file. The load method of the properties class loads the file into memory so that the data can be accessed in a program. A GUI interface was developed which displays the results on screen as they are obtained from the CGE. The aim of this was to demonstrate the structure of the ResourceLookupTable that is held in memory. The displayed results were limited to avoid cluttering the screen with unnecessary data; however the entire XML node will be available to the scheduler when it requests a resource thus providing it will all the necessary information to use the resource.

6 RESULTS

All the information obtained from the system was converted into an XML format and passed to the ResourceLookupTable which filtered this information so only the resources suitable for the current job were included in the lookup table. Once the implementation was complete the system was executed using the MDS server provided by the Globus project. This provided a live environment in which to test the system and therefore it is hoped that the obtained results will more accurately reflect the use of this system. As the system is designed to periodically query the GRID to determine the state of the various resources testing the system was simply a matter of running the application for a period of time so that a series of results could be obtained for later analysis. The results were held in the user's temporary directory as

a series of XML files, each stamped with the date and time to which they correspond. The system was run over a period of three hours to take the required results. From examing the XML result files in the user's temporary directory we can see that there have been a number of results obtained from the MDS server and therefore a means to compare these files was required in order that analysis could be conducted. Due to the fact that the XML files held information coming from the same MDS server and looking at the file size they are all sized between 55 KB and 64 KB, therefore we would expect to find mainly slight variations in these files. As the files held XML data that is structured the same in each file, visual file comparison tools such as that included with Microsoft Visual Source Safe[1] could be used to easily compare the files as illustrated in Figure 2. By comparing the files it is possible to easily spot differences from one file to the next, some of these were the result of variations in the dates when the resources would be valid, but others were of more direct interest to the work in this paper. Mainly changes in the amount of available space as changes such as this could be used by the system to move the current job to another resource. Source safe can be used to compare the files.

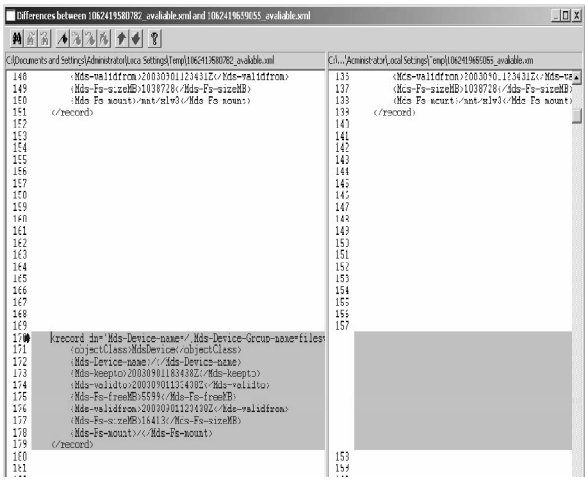

Figure 2: Comparing the result files

When examing using source safe, any blue resources on the left represent a resource that has not been included with the newer set of results. Potentially this means the device may have gone off line and therefore is no longer available for use by the application. Once an analysis of the files had been completed using Visual Source Safe's file comparison tool the results of this analysis can be plotted in a series of graphs which illustrate the observed patterns and characteristics of the results.

[1] http://msdn.microsoft.com/ssafe/

6.1 Available Resources

During tests, files would vary in size. While the majority of files hold 64K of data a large number hold considerably less, obviously a result of containing less information, therefore at that moment in time less results have been returned from the MDS server. This can be confirmed by examining the graph in Figure 3 which plots the number of results returned at a given moment in time.

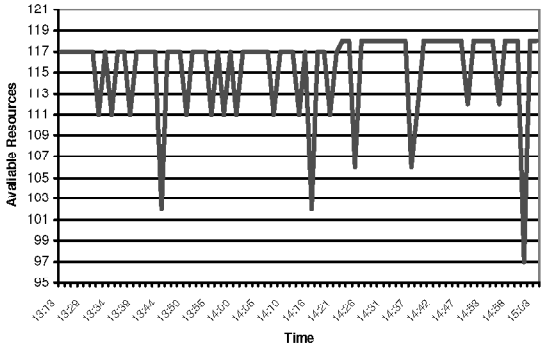

Figure 3: Available resources over time

Figure 3 illustrates the dynamic nature of a CGE in which heterogeneous resources will constantly change as they become over or under utilized, and in which new device come online while existing devices become unavailable due to network failures, system crashes or any number of variables that alter the status of devices.

6.2 Resource Utilization

Alongside simply discovering the available resource attached to the CGE it is anticipated that using the discovered information the system will be capable of making informed decisions upon which resource to use based on past experience.

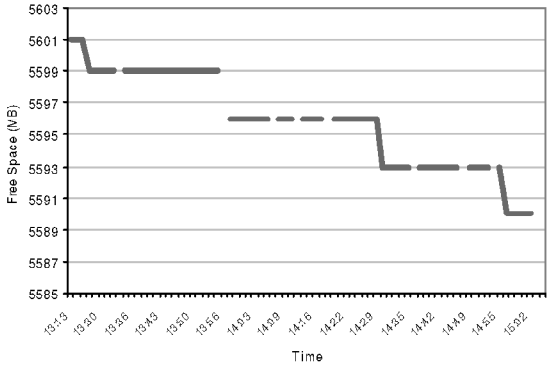

Figure 4: Resource usage of a single resource

During the current experiment the type of resource being queried for was storage therefore tracking the available

storage on the device will give insight into potential tracking utilisation could have in deciding which resource to use. The breaks in the line (in figure 4) correspond to period when the resource was not found in the XML file, in the case of this resource this could be due to a late response to the query request. From studying the XML file the resource has 16413 MB of storage to begin with 5601 MB of this is free space therefore this means that approximately the device has 66% utilization to begin with. The available free space is reduced to 5590 MB by the end of the monitoring however the actual percentage utilization is still approximately 66%.

The current implementation has shown that it is possible to construct a monitor based on the CoG toolkit and the MDS service to monitor the available resources attached to a CGE. This work has built a monitor as a thread that works with a single MDS server, although a commercial implementation would require the use of different MDS servers so that the scheduler is presented with the best possible set of resources to choose from. It is impossible to consider providing the application with a list of MDS servers at compile time as like any resource attached to a network they can be available or unavailable at certain times depending on network conditions, therefore the application will require the ability to dynamically discover the available MDS servers on the GRID. In this implementation the actual monitor was developed as a Java thread as the system was only working with one MDS server. When the system is made to work with multiple servers the monitor will have to be modified so that it becomes an agent and can query multiple MDS servers at once reporting these results back to the client.

7 CONCLUSION

Although the MDS service of the Globus project does provide useful information relating to the state of GRID resources there is a need for more information to be provided about the device particularly relating to accounting information such as the price per megabyte of disc or the price per processing cycle for a processing resource. Standards are important for this type of information as it will facilitate the creation of many different GRID accounting applications all of which would thus be interoperable with one another. It is clear that a CGE consists of many heterogeneous devices which vary in the services they provide and availability. In addition, the period during which these resources can be used varies due to clients using GRID reservation protocols to reserve the device or the resource being used by a client process. Therefore in an environment client programs will require a means to find the available devices and make use of them. An alternative to reservation protocols is useful in case reservation protocols that allow the resources to be booked in advance are priced at a premium rate and thus potentially out of the reach of smaller organisations. Our system allows client applications to discover the available resources at runtime and make use of them concentrating on the area of resource discovery within a CGE. It is successful in discovering information relating to the available devices in a CGE at any moment in time.

REFERENCES

[Arnold99] K. Arnold, B. Osullivan, R. The Jini Specification. The Java Technology Series. Addison-Wesley, June 1999.

[CoG03] CoG Kit Home page, August 2003, http://www unix.globus.org/cog

[Czajkowski98] Czajkowski, I. Foster, N. Karonis, C. Kesselman. A resource management architecture for metacomputing systems. 4th Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[Foster96] I. Foster, C. Kesselman and S. Tuecke. The Nexus approach to integrating multithreading and communication. J. Parallel and Distributed Computing, 37(1):70-82, 1996.

[Foster00] Ian Foster and Carl Kesselman. The GRID: a blueprint for a new infrastructure. Morgan Kaufman, 2000

[Gamma03] E. Gamma, R. Helm, R Johnson, J. Vissides. Design Patterns. Addison Wesley. Dec 2003

[Globus03] Globus Project. http://www.globus.org, 2003

[Gridbus03] The GRIDbus Project, http://www.GRIDbus.org/

[Hardin68] Hardin, G. The Tragedy of the Commons, Science, 162(1968):1243-1248

[Java03] Java Technology & XML. Sun Microsystems. http://java.sun.com/xml. 2003

[Khan03] Khan, M. Vaithianathan, S., Sivoncik, K., Boloni, L. Towards an agent framework for grid computing, International Scientific journal of Computing. Vol. 2, No. 2, December 2003

[Oram01] A. Oram. P2P - Harnessing the Power of Disruptive Technologies. O'reilly, March 2001.

[Laszewski00] G. von Laszewski, I. Foster, J. Gawor, CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance GRIDs. ACM Java Grande Conference, pp. 97–106, San Francisco, CA, 3-5 June 2000.

[Martin00] Dider Martin et. al. Professional XML, Wrox. 2000

[MDS03] Information Services in the Globus Toolkit 3.0. http://www.globus.org/MDS/. 2003

[Rogerson97] D. Rogerson. Inside COM - Microsoft's Component Object Model. Microsoft Press, 1997.

[SAX03] The SAX project. http://www.saxproject.org

[Sun03] Sun Microsystems, Sun Microsystems GRID Technology, 2003, http://www.sun.com/GRID

[XML03] Extensible Mark-up Language (XML), W3C. http://www.w3.org/xml. 2003

Received: Oct. 2004. Accepted: Dec. 2004.