GridARM: Askalon's Grid Resource Management System [⋆]

Mumtaz Siddiqui and Thomas Fahringer

Institute for Computer Science, University of Innsbruck,
Technikerstrasse 13, A-6020 Innsbruck, Austria
{Mumtaz.Siddiqui, Thomas.Fahringer}@uibk.ac.at

Abstract. The emergence of Grid computing has accentuated the need of an adaptable, scalable and extensible resource management system. In this paper we introduce GridARM system which renders the boundaries of resource brokerage, virtual organization wide authorization and advanced reservation, and represents a scalable and adaptive Grid resource management as a middleware infrastructure. The GridARM system provides mechanisms for Grid resource discovery, selection and allocation along with resource requestor and provider interaction. Experiments are presented that demonstrate the effectiveness of our approach.

1 Introduction

With the emergence of distributed computing, and the 'always on' environment of the computing elements; the computing services become scalable, extensible and environment independent. This results in a high performance computational environment composed of diverse resources spanning the entire Internet and multiple administrative domains. The new discipline called *Grid Computing* intends to make high performance computational resources available to anyone. An effective Grid Resource Management System (GRMS) is required for the provisioning and sharing of resources while keeping autonomy of their environment and geographical location.

In contrast to traditional resource management system, GRMS has to balance global resource sharing with local autonomy, by dealing with heterogeneous, shared and variant resources distributed under different trust domains, addressing issues of multiple layers of schedulers and working with system participants having inconsistent performance goals and assorted local and global policies.

In the Grid computing literature, Grid job scheduling is represented and treated as part of the Grid resource management, therefore in most of the existing Grid enabled systems, meta scheduling is integrated with resource management and the terms *Grid job scheduling* and *Grid resource management* are used interchangeably. Now the Grid computing has been evolved enough to redefine and redesign its components so that they can be used as self comprised building blocks in GRMS. Resource broker is one of the important GRMS components,

[⋆] This research is supported by the Higher Education Commission of Pakistan which is manual or semi manual in existing systems. An automatic resource broker is essential for a stable and successful Grid computing infrastructure.

A GRMS must provide *Resource discovery and selection mechanism* which performs persistent resource state and capacity checking, by discovering and matching resources, *Capability check mechanism* which performs resource selection based on dynamic information, *Resource allocation* with advance reservation and co-allocation and finally *Resource requester and provider interaction* for negotiation and notifications.

To our knowledge no widely deployed single GRMS supports these functions. Attribute based resource description and resource matching available in existing systems is unsuitable for ever evolving Grids. Virtual Organization (VO) wide authorization, advance reservation and Co-allocation are still illusions for the Grid users. We propose a new approach to flexible resource management for Grid computing system which provides the management functions mentioned above. The goal of our work is to provide an effective, efficient, extensible and adaptive GRMS based on off-the-shelf technologies while making it capable to adapt new emerging technologies.

This paper presents a design architecture and work-in-progress prototype implementation of *GridARM* (Askalon's [14] Grid Resource Management) System, a new architecture for Grid resource management, resource control and resource provisioning across sites and administrative domains. It provides automatic resource brokerage, VO-wide fine-grained authorization, advanced reservation and negotiation between a potential client and resource provider. The automatic broker was necessary not only because of its usability, efficiency and low cost but also because users don't have time to make selection between alternative choices. VO-wide authorization and user profiling mechanism reduces involvement of local site administrators and even the user itself.

The rest of the paper is as follows. In Section 2, we describe general Grid resource management architecture and define basic mechanisms to provide an automatic resource brokerage system. Section 3 is about the client-GridARM interaction mechanism. In Section 4, we described our experiences about the proposed system, and examined its performance in a networked environment. Related work is presented in Section 5. Finally we summarize our conclusion about the proposed system and discuss future work in Section 6.

2 GridARM Architecture

The GridARM system is dynamically extensible, scalable and adaptive in which new protocols and tools can easily be integrated without suffering from system downtime and expensive code reorganization. In contrast to existing work which is based on manual brokerage we propose an automated brokerage in this system. This automation is required especially for Grid enabled workflows and execution environments where the brokerage process acts as a middle tier between Meta-scheduler and other Grid enabled components like Grid enabled resources and services. The brokerage process is responsible to discover and allocate suitable resources for the Meta schedulers.

We are developing *GridARM* system which is WSRF [11] complaint. The system consists of four persistent and distributed Grid-enabled services called Discovery, Authorization, Reservation and Broker.

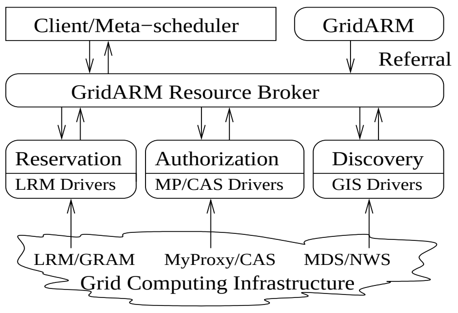


Fig. 1. GridARM system architecture.

As shown in the Fig. 1, the Broker service is a gateway to the GridARM sytem. It is a configurable and customizeable WS-Resource which works with one or more other GridARM services. An important feature of the Broker is its ability to recursively discover Grid resources by interacting with the other distributed GridARM brokers.

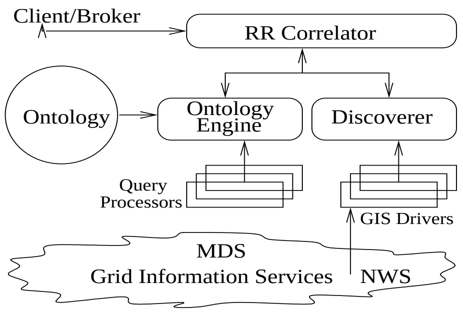
The *Discovery service* provides resource discovering and matching capability, mainly to the Broker. It can be configured with one or more Grid Information Services (GIS).

Authorization and *Reservation* services provide resource authorization and advance reservation capabilities respectively. Authorization is based on the Grid Security Infrastructure (GSI) [3] and works in coordination with My-Proxy [7] and Community Authorization Service (CAS) [2]. The Reservation service provides capabilities like advanced reservation and negotiation for reservation with the resource provider, based on time and cost model and other constraints set by the both parties.

Each *GridARM service* is configured to have one or more drivers to the resource specific modules like GIS service. In the following sections we describe GridARM services in detail.

2.1 Discovery Service

A resource discovery mechanism mainly provided to the *Broker* service is based on a set of information and monitoring services. It discovers and provides an optimal resource *ensemble* along with solicited specification congregated from the underlying information services. We are currently working on a *Grid resource description language*, a language to be used for describing Grid resources in an ontological way. The internal structure of a discovery service is illustrated in Fig. 2. It consists of a *Request-Resource Correlator (RRC)*, *Ontological Engine (OE)* and *Resource Discoverer (RD). Correlator* receives a request, checks its integrity, correlates it with the resources and returns the result back. It interacts with *OE* and the *RD* for request transformation and resource discovery. A client can subscribe in the discovery service by registering resource requirements or preferences in the *RRC* and receives notification from it when a matching resource joins the Grid and observed by the discovery service.



```
<ResourceRequest>
 <resources>
  <totalNodes constraint="exact"> 15 </totalNodes>
  <constraints>
   <cpuCount constraint="min">4</cpuCount>
   <cpuAvailable constraint="min">2.5</cpuAvailable>
   <osType flavor="any">Linux</osType>
   <platform>sparc</platform>
  </constraints>
 </resources>
</ResourceRequest>
```

Fig. 2. A: Internal structure of a discovery service. **B**: A simple resource request with both static and dynamic attributes.

Ontological Engine selects appropriate *Query Processor (QP)* and transforms the request into resource *filters* for the registered information services, based on which resources are discovered and selected. The transformation is done in two steps: First, the compound request is split into multiple but simple requests, for example by separating static and dynamic attributes of a request and making two different GIS-specific requests. Second, it transforms requests from generic format to GIS specific *filters*. For instance, a generic request given in Fig. 2 is transformed into LDAP filter for the MDS2 [4] driver.

The *RD* component discovers resources and their specification based on the *filters* provided by *OE* and congregates all found resources along with their specification into a unique GIS-independent generic format. Finally it returns the result back to the *Correlator*.

Discoverer drivers and *query processors* are loaded dynamically based on the GIS and query *type* respectively. The default GIS type is *mds2* and query type is *generic*. A client can make a query for resources based on both static and dynamic attributes. For instance, a resource request given in Fig. 2, contains both static and dynamic attributes and discovery service uses MDS to collect static information like *OsName* and *Platform* and NWS [5] to collect dynamic information of resources such as $AvailableCpu$.

2.2 Reservation Service

Reservation is an undertaking by the system that an application will receive a certain level of service from its resources. The reservation service provides this functionality by supplying *instant* as well as *advanced reservation* of underlying Grid-enabled resources. The architecture is flexible enough to work with different local reservation managers for example Maui [6]. But our aim is to provide a new VO wide reservation manager which is consistent with other Globus Toolkit (GT) based components like CAS [2]. This service is used to interact with a single resource, whereas Co-allocation of multiple resources is handled by the Broker service which also acts as a *Co-allocation manager*.

The reservation service provides high level methods to *create*, *modify*, *bind*, *cancel*, monitor and *verify* a resource *reservation* instance. Verification is required to be performed before acquiring a reserved resource by a job submission component. Also, the service can be used to look ahead for the advanced reservation of a resource by employing its *lookahead* method. The Local Reservation Managers (LRM) can be registered and managed dynamically.

The essential attributes of a reservation instance are *LRM contact*, *reservation mode*, *start time*, *end time*, *duration* and *constraints*. The reservation mode can be a *single phase* or a *two-phase committable.* In two-phase mode, once a reservation is created, it remains on hold for a configurable *duration.* If reservation is not committed within that time interval, then it is cancelled automatically. The reservation, which is not committed with in specified *time-frame*, represents a soft allocation of resource as described in Section 2.4. The optional *end time* attribute can also be provided for flexibility. The reservation is accepted if it can be started any time after *start time* and finished any time before *end time.*

A special *Constraint-based Advanced Reservation (CAR)* can be created based on resource constraints instead of hard coded resource manager contact. In CAR, the LRM contact *linking* is deferred until *bind* time. Reservation service ensures that minimum required resources which fulfill the CAR's constraints should be available during that *time frame.* In this way the required QoS is ensured.

A *Ticket*, which is granted by the service after making a reservation, is a reference to a reservation instance. It embraces a unique *reservation id*, user's *security principal* and resource *access point.* Once a reservation has been made, all future interactions, like monitoring for its status, modification, cancellation and resource acquisition can only be done by producing a valid *ticket.* Before performing any task, a user *credential* is produced or retrieved from the Authorization service. The reservation is not possible if a user possesses invalid credential or a given policy does not permit a reservation. A policy is described in the form of constraints and used with authorization information to provide enhanced quality of service. One can specify a list of users or groups allowed to use a reservation ticket in subsequent operations. The integrity is ensured by signing the policy with trusted entity.

2.3 Authorization Service

The Globus system lacks a middleware-based authorization. It uses local resource mechanism for authorization by mapping a Grid user to a local identity which also serves as an access control check. This scenario has several shortcomings such as scalability, lack of expressiveness, and consistency between the policies of different sites. In order to overcome these shortcomings GridARM proposes an authorization mechanism, in which, the Grid users and sites are registered in a middleware service, which maintains user profiles, proxy credentials, policies and preferences. The Grid sites are registered by creating a local identity representing the VO or community.

This service grants authorization to a user by verifying its access rights for a particular *resource ensemble* and provides a *restricted proxy credential* to the user. A restricted proxy credential can only be used during the reservation time-frame. Our authorization service uses Globus My-Proxy [7] as a credential repository, and we plan to integrate Community Authorization Service (CAS) [2] in our approach.

2.4 Broker Service

The *Broker* service works as a gateway to the *GridARM* system. It makes an efficient and smart use of the other services, which can be registered and managed dynamically. It also works as a *Co-allocation manager* and performs advance reservation of multiple resources on request. Its role in the overall system is illustrated in Fig. 1. High level interfaces are provided for resource selection, allocation and management. These interfaces include methods like *select*, *allocate*, *confirm* and *release* etc. Selection operation results in a *resource ensemble* based on the resource request, whereas allocation operation results in a *reservation ensemble* or reservation *ticket*. The input to the broker, provided mainly by Grid scheduler, is examined and an appropriate and optimal operation is performed.

The allocation of resources can be a result of an *interactive transaction* by following *select-allocate-confirm* cycle in steps, or it can be an *atomic transaction* by making a direct confirmed reservation. An allocation of resources without confirmation is a *soft allocation*. A soft allocation is one in which allocated resources will be available to new clients only if (1) they are explicitly released or (2) they are not used within certain configurable *duration* of time. Once reservation is confirmed it becomes a *hard allocation* that means the allocated *resource ensemble* remain dedicated to a client during the reservation *timeframe*.

If a broker could not find suitable resources, it can refer its clients to another broker service, or it can be configured to work in a recursive mode to retrieve required resources by interacting with the *remote* brokers.

Once a reservation ensemble is created, the Broker instantiates a resource *Ensemble Manager (EM)* on one of least loaded GridARM-enabled hosts. *EM* is responsible for further coordination and negotiation with client on behalf of resource providers. It provides comprehensive functionality for editing and managing a reservation ensemble. Also *EM* monitors its members while they are being used by the client and ensures that resources are working according to the constraints specified at the time of reservation. A node failure, if occurs, is notified to the job submission system.

The GridARM system works based on GSI [3] provided by the Globus. A client can interact with system via frontend broker service by providing its own proxy credential supplied directly or through My-Proxy [7]. If an inter-VO-referral based recursion is performed, then it would be possible that a requesting user is not part of a referred VO. In this case user credentials are replaced with referral or 'remote' broker's credential and a chained authorization is performed to make it possible. The management interfaces of GridARM services are implemented with message level security, so that an unauthorized Grid user could not make changes in the configuration.

3 Interaction Mechanism

Advanced reservation and Co-allocation of a *resource ensemble* is a multi transactional process, which is simplified by providing a *Proxy Resource Ensemble(PRE)* in the response of a resource request. The broker *execute*s a request by selecting a *resource ensemble*, instantiating an *Ensemble Manager (EM)*, and returning back a *proxy (PRE).* The *PRE* being a mobile agent, is downloaded to a client machine and used as a stub of the resource ensemble for further interaction with the *GridARM* system. It hides authorization mechanism and optionally physical resources from the client. This is done by providing a *login mechanism*, in which the user along with reservation *ticket* is verified, and the *user credential* is replaced with *restricted community credential.* The client presents community credential to the resource ensemble before submitting a job. We plan to integrate a policy evaluation mechanism in the PRE in order to provide a fine-grained user authorization by the PRE.

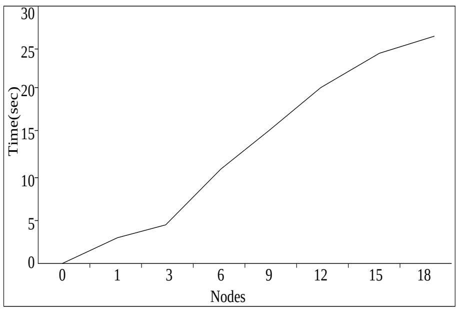
4 Experiments

The Globus toolkit provides a decentralized scheduling model in which new components can be integrated. An unofficial release of GT4 which consists of core implementation of WSRF [11] was introduced in the beginning of 2004. WSRF is a new model on which Grids are to be built. The GridARM system is WSRF complaint and uses mechanisms like subscription/notification and service lifetime management. We have deployed the system in ZID-Grid, University of Innsbruck, which consists 13 Grid sites, one with 15 Solaris machines and 12 with 141 Linux PC boxes. GT2 is installed on all Grid sites whereas GT4 core is installed only on Solaris machines. Apart from site specific services like GRAM [1], we have installed NWS [5] and MDS2 [4] with the Glue schema. Both MDS and NWS cover all ZID-Grid sites. All the machines involved in the experiment were located on a lightly loaded network with a maximum latency between two computers of about 2 milliseconds.

A resource request can be a simple or compound request. A *simple request* could be an attribute-based request for the selection of resources or a reservation request for already selected resource. A *compound request* is one in which multiple operations are requested atomically. In the following sections we describe different experiments conducted in the deployed Grid infrastructure.

4.1 Atomic transaction

A compound request shown in Fig. 3 is made to the system for resource allocation. The request consists of both resource and reservation description. In this scenario, the broker service performs resource *selection*, *authorization*, *allocation* and *confirmation* cycle as a *single* transaction. To determine the cost of requested operation, we timed a series of requests varying both the total number of required resources and their attributes. We measured the time for the request

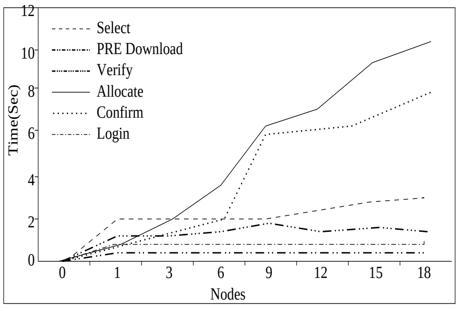


```
<ResourceRequest>
  <type> compound </type>
  <resources negotiation="no">
    <constraints>
      <cpuCount constraint="min">3</cpuCount>
      <cpuCurrent constraint="min">1.5</cpuCurrent>
      <osType flavor="any">Linux</osType>
      <platform>intel</platform>
    </constraints>
  </resources>
  <reservation negotiation="yes">
    <startTime format="YYMMddhhmm" constraint="exact">
      0410151230
    </startTime>
    <duration unit="minutes">150</duration>
  </reservation>
</ResourceRequest>
```

Fig. 3. A: GridARM system latency for resource reservation, time taken for atomic transactions for the given number of nodes/CPUs. **B**: Atomic transaction: A compound request for a resource ensemble selection and confirmed allocation.

by starting a timer in the client program immediately before invoking the broker and then stop this timer on successful login to the resource ensemble through proxy *PRE* (See Section 3). The result of this experiment is shown in Fig. 3. The graph shows how the time for request brokerage varies as the number of resources changed. A further breakdown of the time spent in different GridARM operations is given in Fig. 4, which shows the cost of each operation.

4.2 Interactive transaction



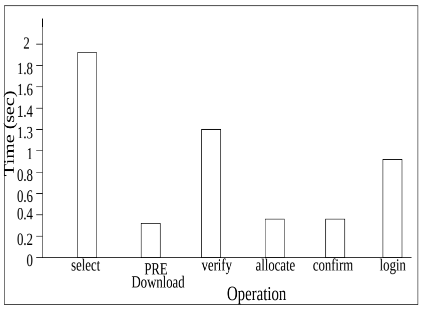


Fig. 4. A: Time variation of GridARM system functions. **B**: Breakdown of times spent in processing a resource allocation request for a single resource.

In this experiment, the *compound request* shown in Fig. 3, is broken down into two requests, and the brokerage is performed in steps by implying *execute*, *verify*, *allocate*, *confirm* and *login* functions in a sequence by the client. The interactive transaction is useful in case when a client wants to perform some intermediate tasks. A simple use case could be as follows: A Metascheduler first reserves the resource ensemble and then confirms after interacting with a performance predictor, that may evaluate a Grid resource in the meantime. A client can also call *lookahead* function for the reservation of the resource ensemble before making a confirmed reservation.

A breakdown of the time spent in different system functions is given in Fig. 4. The authorization (*verify* + *login*) and discovery (*select*) services are very consistent and economical. Most of the time is consumed by *allocate* and *confirm* functions of the reservation service. This is due to the fact that currently a separate request is made for each resource in the resource ensemble. We plan to enhance the service by adding the functionality in which an entire resource ensemble could be handled with a single request. As shown in the Fig. 4(right), *allocate* and *confirm* functions collectively take less than one second, therefore there will be a significant improvement in the system performance after having the enhanced functionality.

5 Related Work

In the domain of GRMS, numerous projects and tools are available, but most of them do not provide the required level of resource management. This pervasive domain needs to split down further in more self contained and adaptable sub domains. Most of the existing Grid enabled systems try to address resource brokerage, job scheduling and monitoring under the same integrated scenario. It works, but it's not scaleable and adaptable. The resource broker in the Globus system is missing. A few Grid systems like Condor [8], Legion [9], GridLab [16], European Data Grid [10], Nimrod-G [15] and Maui [6] address GRM but the broker is not a well divulged and concrete module. Also none of these systems addresses resource management as a mechanism of the Grid middleware, in which distributed resource brokerage, community-based authorization and advanced reservation are consistent with each other.

A distributed resource management architecture that supports advance reservation and Co-allocation is described in [12] but the modification proposed in the local resource management is an overhead. Ontology based resource matching proposed in [13] simplifies resource matching, but community based authorization and reservation has not been addressed. The Global Grid Forum (GGF) is actively working on devising new standards in different areas of resource management. The GridARM system will adopt GGF standards once fully specified.

6 Conclusion and Future work

Unleashing the power of Grid infrastructures is a complex and tedious task without a sophisticated resource management system. The focus of this paper is to render the boundaries of resource brokerage, community wide authorization and advanced reservation mechanism. The paper proposes a modular and dynamically extensible Grid resource management architecture, which fills the gap between Meta-scheduler and the Grid infrastructure. The GridARM system makes the use of the Grid resources simple and efficient with the help of VO-wide authorization and reservation mechanism. Our aim is to make the system fully consistent with the Grid forum recommendations. We are evaluating different related technologies which can be exploited to make the system more functional and consistent with emerging web technologies.

The GridARM system automates the process of the Grid resource management, but its own management and VO-wide deployment is manual. The plan for the future enhancement of the system is to make it fully automated.

References

1. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. *A Resource Management Architecture for Metacomputing Systems.* Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, po. 62-82, 1998.
2. L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke. *A Community Authorization Service for Group Collaboration.* IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2001.
3. Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. *A Security Architecture for Computational Grids.* In Fifth ACM Conference on Computers and Communications Security, November 1998.
4. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. *Grid Information Services for Distributed Resource Sharing.* Tenth IEEE International Symposium on High-Performance Distributed Computing(HPDC-10), IEEE Press, Aug 2001.
5. Rich Wolski, Neil Spring, and Jim Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing* Future Generation Computing Systems Journal, Vol 15, 5-6, pp. 757-768, Oct. 1999.
6. *The Maui Scheduler* home page. http://maui-scheduler.mhpcc.edu
7. J. Novotny, S. Tuecke, V. Welch. *An Online Credential Repository for the Grid: MyProxy.* Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
8. *Condor project homepage.* http://www.cs.wisc.edu/condor/
9. Steve Chapin, Dimitrios Katramatos, John Karpovich, Andrew Grimshaw. *The Legion Resource Management System.* Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99)
10. B. Sagal. *Grid Computing: The European DataGrid Project.* In IEEE Nuclear Science Symposium and Medical Imaging Conference Lyon, France, October 2000
11. *Web Services Resource Framework.* http://www.globus.org/wsrf/
12. K. Czajkowski, I. Foster, and C. Kesselman. *Resource Co-Allocation in Computational Grids.* In Proc. of the 8-th IEEE Int'l Symp. on High Performance Distributed Computing, pages 219-228, Redondo Beach, CA, USA, July 1999
13. H. Tangmunarunkit, S. Decker, and C. Kesselman. *Ontology-based Resource Matching in the Grid–The Grid meets the Semantic Web.* Second International Semantic Web Conference, Sanibel-Captiva Islands, Florida, USA, Oct. 2003
14. Thomas Fahringer *ASKALON: A Programming Environment and Tool Set for Cluster and Grid Computing* Institute for Computer Science, University of Innsbruck. http://dps.uibk.ac.at/askalon/
15. R. Buyya, D. Abramson, and J. Giddy. *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, HPC ASIA'2000, China, IEEE CS Press, USA, 2000
16. Allen, G., Davis, K., et al. (2003). *Enabling Applications on the Grid: A GridLab Overview* In International Journal of High Performance Computing Applications: Special Issue on Grid Computing, August 2003.