An Economy Driven Resource Management Architecture for Global Computational Power Grids

Rajkumar Buyya, David Abramson, and Jonathan Giddy‡

School of Computer Science and Software Engineering
Monash University
Caulfield Campus, Melbourne, Australia

‡CRC for Enterprise Distributed Systems Technology
University of Queensland
St. Lucia, Brisbane, Australia

Email: {rajkumar, davida, jon}@csse.monash.edu.au

Abstract

The growing computational power requirements of grand challenge applications has promoted the need for linking high-performance computational resources distributed across multiple organisations. This is fueled by the availability of the Internet as a ubiquitous commodity communication media, low cost high-performance machines such as clusters across multiple organisations, and the rise of scientific problems of multi-organisational interest. The availability of expensive, special class of scientific instruments or devices and data sources in few organisations has increased the interest in offering a remote access to these resources. The recent popularity of coupling (local and remote) computational resources, special class of scientific instruments, and data sources across the Internet for solving problems has led to the emergence of a new platform called "Computational Grid".

This paper identifies the issues in resource management and scheduling driven by computational economy in the emerging grid computing context. They also apply to clusters of clusters environment (known as *federated clusters* or *hyperclusters*) formed by coupling multiple (geographically distributed) clusters located in the same or different organisations. We discuss our current work on the Nimrod/G resource broker, whose scheduling mechanism is driven by a user supplied application deadline and a resource access budget. However, current Grid access frameworks do not provide the dynamic resource trading services that are required to facilitate flexible application scheduling. In order to overcome this limitation, we have proposed an infrastructure called GRid Architecture for Computational Economy (GRACE). In this paper we present the motivations for grid computing, resource management architecture, Nimrod/G resource broker, computational economy, and GRACE infrastructure and its APIs along with future work.

1. Introduction

The concept of grid computing is gaining popularity with the emergence of the Internet as a medium for global communication and the wide spread availability of powerful computers and networks as low-cost commodity components. The computing resources and special class of scientific devices or instruments are located across various organizations around the globe. These resources could be computational systems (such as traditional supercomputers, clusters [5], SMPs, or even powerful desktop machines), special class of devices (such as sensors, radio telescopes, satellite receivers), visualization platforms, and storage devices. A number of applications need more computing power than can be offered by a single resource or organisation in order to solve them within a feasible/reasonable time and cost. This promoted the exploration of logically coupling geographically distributed high-end computational resources and using them for solving large-scale problems. Such emerging infrastructure is called *computational (power) grid* [18], analogous to electric (power) grid, and led to the popularization of a field called *grid computing*. It has been predicted that the global computational grids are expected to drive the economy of the 21st century similar to the electric power grid that drove the economy of the 20th century.

Computational grids are expected to offer dependable, consistent, pervasive, and inexpensive access to high-end resources [18] irrespective of their physical location and the location of access points. A number of projects worldwide are actively exploring the development of grid computing technology. They include Globus [17], Legion [25], NASA Information Power Grid [29], NetSolve [9], Ninf [32], AppLes [7], Nimrod/G [1], DISCWorld [13], and Unicore [2]. In [3], all these grid systems have been discussed.

Although wide-area distributed supercomputing has been a popular application of grid computing, there are a number of other applications that can benefit from it. They include collaborative engineering, high-throughput computing (large-scale simulation and parameter studies), remote software access, data-intensive computing, and on-demand computing. However, our focus is on the use of the grid for solving supercomputing and high-throughput computing applications, in particular.

Due to the use of geographically distributed multi-organizational resources, the grid computing environment needs to dynamically address issues involved in inter-domain resource usage and should have the following features [6] [16]:

- Flexibility and extensibility
- Domain autonomy
- Scalability
- Single global name space
- Ease of use and transparent access
- High performance
- Security
- Management and exploitation of resource heterogeneity
- Interoperability with multiple systems
- Resource allocation or co-allocation
- Fault-tolerance
- Dynamic adaptability
- Economy of computation

A number of middleware systems including Globus have addressed some of the above issues. In [11], the Globus developers have addressed the five challenging resource management problems introduced by computational grids: site autonomy, heterogeneous substrate, policy extensibility, resource allocation or co-allocation, and online control. The sixth challenging resource management problem that drives our work is “economy of computations”. The resource management architecture presented in this paper is driven by the concept of a computational economy and the necessary enabling middleware infrastructure. Importantly, our work can be combined with existing middleware systems such as Globus, to produce an environment that addresses all of the six challenges produced by production oriented computation grids.

The Nimrod/G resource broker, a global resource management and scheduling system for computational grid, built using Globus services has been discussed in [1][6]. It supports deadline and cost-based scheduling mechanism, but the costing mechanism is currently static. We have found that the Globus metacomputing toolkit does not offer services for trading resources dynamically. This limitation is overcome by our proposed GRid Architecture for Computational Economy (GRACE) middleware infrastructure that co-exist with Globus, and Nimrod/G can use for trading resources to support dynamic scheduling capability.

The focus of this paper is on economy driven resource management architecture for grid computing. It addresses the first five resource challenges through the use of Globus middleware services and the sixth challenge through GRACE infrastructure. Our work is concerned with the resource discovery, brokering and economy of computations, resource acquisition, scheduling, staging data and programs, initiating computations, adapting to changes in the grid status, and collecting results.

The remaining sections of this paper discuss resource management models, related work, motivations for an economy driven resource management system and its architecture, GRACE infrastructure, an architecture for the Nimrod/G resource broker that supports deadline based scheduling and dynamic resource trading using Globus and GRACE services. The summary and future work are presented at the end.

2. Resource Management Structures

The architectural model of resource management systems is influenced by the way the scheduler is structured. The structure of scheduler depends on the number of resources on which jobs and computations are scheduled, and the domain in which resources are located. Table 1 shows scheduler structural models for different combinations of resources and the location of their existence. Primarily, there are three different models for structuring schedulers:

- **Centralized scheduling model:** This can be used for managing single or multiple resources located either in a single or multiple domains. It can only support uniform policy and suits well for cluster management (or batch queuing) systems such as Condor [10], LSF [26], and Condine [23]. It is not suitable for grid resource management systems as they are expected to honor (local) policies imposed by resource owners.
- **Decentralized scheduling model:** In this model schedulers interact among themselves in order to decide which resource should be applied to the jobs being executed. In this scheme, there is no central leader responsible for scheduling, hence this model appears to be highly scalable and fault-tolerant. As resource owners can define the policy that schedulers can enforce, the decentralized scheme suits grid systems. However, because the status of remote jobs and resources is not available at single location, the generation of highly optimal schedule is questionable! This model seems difficult to implement in the grid environment, as domain resource owners do not agree on a global policy for resource management.

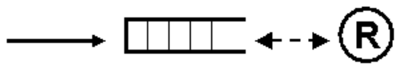
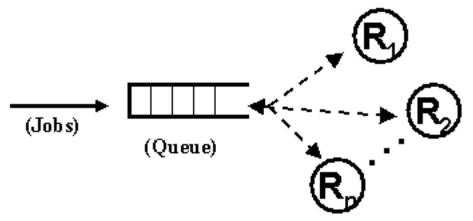
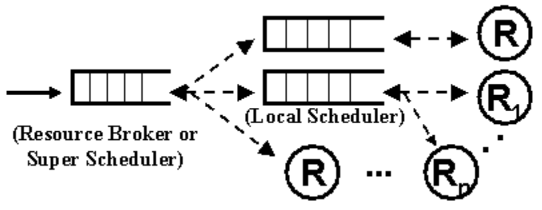
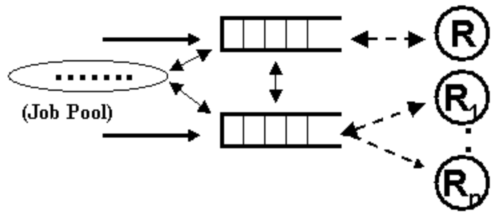
Scheduler Model	Scheduler Architecture	Example System
Centralized (Single Resource)		Unix, Linux, Windows OS
Centralized (Multiple Resources) (Single/Multiple Domains)		Cluster systems such as LSF, Codine, Nimrod, Condor
Hierarchical (Multiple Domains)	 (Resource Broker or Super Scheduler) (Local Scheduler) R R1 R ... Rp	Multi-clusters or Grid Systems such as AppLes, Nimrod/G, HTB, Legion, Condor
Decentralized (Self coordinated or Job Pool) (Single/Multiple Domains)		Cluster systems like MOSIX follows simple model. Grid systems can follow, but it is complex to realize.

Table 1: Scheduling Structure Alternatives.

- **Hierarchical scheduling model:** This model fits for grid systems as it allows remote resource owners to enforce their own policy on external users. This model looks like a hybrid model (combination of central and decentralized model), but appears more like centralized model and therefore suits grid systems. Our resource management architecture follows this model. The scheduler at the top of the hierarchy is called super-scheduler/resource broker that interacts with local schedulers in order to decide schedules.

3. Related Work

A number of resource management architectures have been proposed at the Grid Forum (GF) [21] Scheduling Working Group. The first proposal tries to explicitly capture almost all features supported by resource management systems currently being developed [8]. The second proposal comprises a scheduling *tile* with three parts: a "mapper", a "commit agent" and a "deploy agent" [30]. These tiles could be layered in a hierarchical manner, so that one can have tiles—each to represent the local systems, a tile higher up as a system-level scheduler, and so on. The third proposal, Abstract Owner (AO) model, emphasizes *order and delivery* approach and captures real world model, however, currently there are no software systems that support this model [12]. Our resource management architecture captures the essence of all of them and presents in a simple, realistic, and easily realizable manner. Based on the reply [22] to our proposal for including computational economy (economy driven resource management/scheduling model) in the charter, it is expected that the GF scheduling group will address it in future! The infrastructure supporting the distributed accounting model discussed in the GF account management working group draft [33], can become a substrate for our work.

The existing systems have addressed computational economy in a different context: Mariposa, a distributed database system, supports economy in database query processing [27]. Rexec, remote execution environment, is targeted for clusters where resource share is allocated based on the relative economical value that the user assigns to the job [4]. The grid systems such as Globus[19], Legion [25], Netsolve [9], AppLes [7], and Condor [10] neither offer resource trading services nor support job scheduling with economy of computations. JaWS [24] follows an economy-based web-computing model where resource owners (desktop users) visit a URL to contribute their resources.

4. Why Computational Economy?

The grid is constructed by coupling resources distributed across various organizations and administrative domains and may be owned by different organisations. The need for an economy driven resource management and scheduling system comes from the answers to the following questions:

- What comprises the Grid?
- What motivates one to contribute their resource to the Grid?
- Is it possible to have access to all resources in the Grid by contributing our resource?
- If not, how do we have access to all Grid resources?
- If we have access to resources through collaboration, are we allowed to solve commercial problems?
- If we gain access to Grid resources by paying money, do resource owners need to charge the same or different price for other users?
- Is access cost the same for peak and off-peak hours?
- How can resource owners maximize their profit?
- How can users solve their problems within a minimum cost?
- If the user relaxes the deadline by which results are required, can solution cost be reduced?

The motivations or incentives for contributing resources towards building grids, to date, has been driven by public good, prizes, fun, fame, or collaborative advantage. This is clearly evident from the construction of public or research test-beds such as, SETI@Home [31], Distributed.net [14], DAS [15], and GUSTO[20]. The computational resource contributors to these test-beds are mostly motivated by the aforementioned reasons. The chances of gaining access to such computational test-beds for solving commercial problems are low. Furthermore, contributing resources to a testbed does not guarantee access to all of the other resources in the testbed. For example, although we are part of the GUSTO testbed, we do not have automatic access to all of its resources. Unless we have some kind of collaboration with contributors, it is difficult to get access to their resources. In this situation, we believe that a model that encourages resource owners to let their resources for others use is computational economy – wherein users are charged for access at a rate that varies with time. This necessitates the need for a mechanism where one can buy compute power on-demand from computational grids or resource owners. As both resource owners and users want to maximize their profit (i.e., the owners wish to earn more money and the users wish to solve their problems within a minimum possible cost), the grid computing environment needs to support this economy of computations.

In order to push the concept of grid into mainstream computing, we need a mechanism that motivates owners to contribute their machine (idle) resources. One of the best mechanisms for achieving this is supporting the concept of computational economy in building and managing grid resources. It allows resource owners to earn money by letting others use their (idle) computational resources for solving their problems. In such a production oriented (commercial) computational grid, the resource owners' act as sellers and the users act as buyers. The pricing of resources will be driven by demand and supply and is one of the best mechanisms to regulate and control access to computational resources.

The grid resource management systems must dynamically trade for the best resources based on a metric of the price and performance available and schedule computations on these resources such that they meet user requirements. The grid middleware needs to offer services that help resource brokers and resource owners to trade for resource access.

The benefits of economy-based resource management include the following:

- It helps in building large-scale computational grid as it motivates resource owners to contribute their idle resources for others to use and profit from it.
- It provides fair basis for access to grid resources for everyone.
- It helps in regulating the demand and supply.
- It offers an incentive for users to back off when solving low priority problems and thus encourages the solution of time critical problems first.
- It removes the need for a central coordinator (during negotiation).
- It offers uniform treatment to all resources. That is, it allows trading of everything including computational power, memory, storage, network bandwidth/latency, and devices or instruments.
- It helps in developing scheduling policies that are user centric rather than system centric.
- It offers an efficient mechanism for allocation and management of resources.

- It helps in building a highly scalable system as decision-making process is distributed across all users and resource owners.
- Finally, it places the power in the hand of both resource owners and users—they can make their own decisions to maximize the utility and profit.

5. Economy driven Grid Resource Management Architecture

The resources that are coupled in grid computing environment are geographically distributed and different individuals or organizations own each one of them and have their own access policies, cost, and mechanisms. The resource owners manage and control resources using their favorite resource management and scheduling system (called local scheduler) and the grid users are expected to honor that and make sure they do not interfere with resource owners' policies. They may charge different prices for different users for their resource usage and it may vary from time to time. The global resource management and scheduling systems (e.g., Nimrod/G [1]), popularly called grid schedulers or meta-schedulers, coordinate the user access to remote resources in cooperation with local schedulers (e.g., Condor [10], Codine/GRD [23] and LSF [26]) via grid middleware services (e.g., Globus [19]). Traditionally, most of the schedulers follow system centric approach (e.g., they just care about system performance) in resource selection and often (completely) ignore the user requirements (e.g., resource access cost). In order to overcome this problem, we proposed an economy-based approach for grid resource management and scheduling system architecture shown in Figure 1. When the user submits an application for execution, they expect that the application be executed within a given deadline and cost. They also need a means for trading off the cost and the deadline. These requirements appear complex, but under a computational economy they simplify the scheduling problem and reduce the complexity involved in the design and development of grid schedulers. There is no single perfect solution that meets all user requirements, hence the requirements (schedulers) are tailored for each class of applications.

The following are the key components of our resource management system:

- User Applications (sequential, parametric, or parallel applications)
- Grid Resource Broker (a.k.a., Super Scheduler, or Global Scheduler)
- Grid Middleware
- Local Resource Manager (Scheduler) such as Condor and LSF

Grid Resource Broker (GRB)

The resource broker acts as a mediator between the user and grid resources using middleware services. It is responsible for resource selection, binding of software (application), data, and hardware resources, initiate computations, adapt to the changes in grid resources and present the grid to the user as a single, unified resource. The components of resource broker are the following:

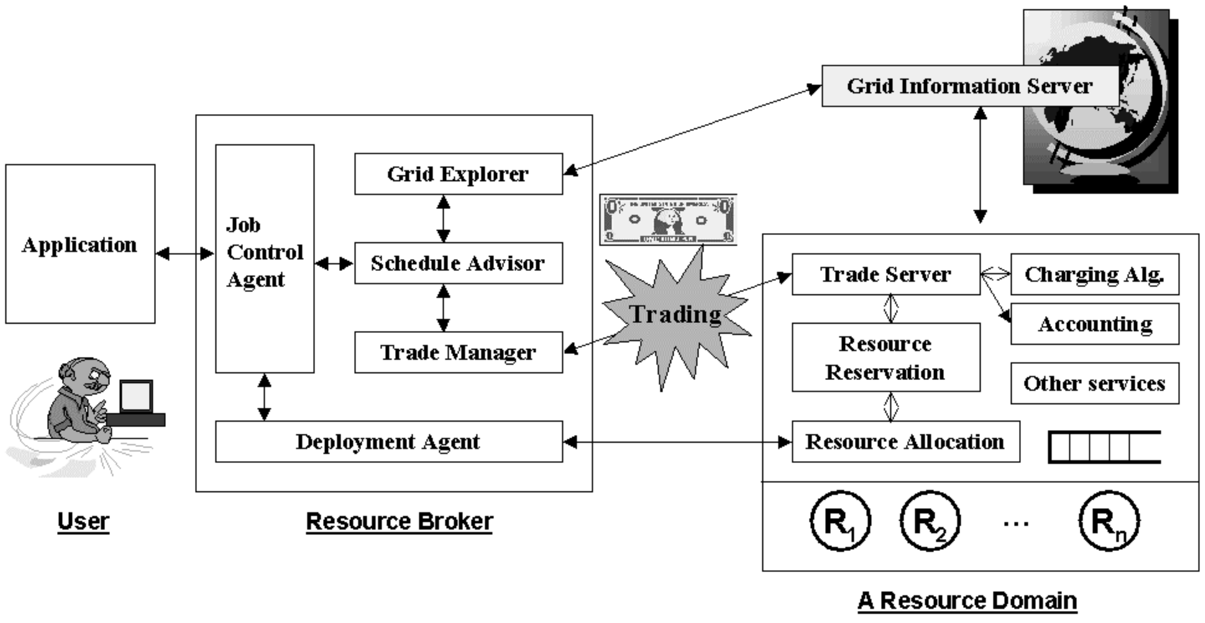


Figure 1: Economy driven Grid Resource Management Architecture.

- **Job Control Agent (JCA):** This component is a persistent central component responsible for shepherding a job through the system. It takes care of schedule generation, the actual creation of jobs, maintenance of job status, interacting with clients/users, schedule advisor, and dispatcher.
- **Schedule Advisor:** This component is responsible for resource discovery (using grid explorer), resource selection, and job assignment. Its key function is to select those resources that meet user requirements such as meet the deadline and minimize the cost of computation while assigning jobs to resources.
- **Grid Explorer:** This is responsible for resource discovery by interacting with grid-information server and identifying the list of authorized machines, and keeping track of resource status information.
- **Trade Manager:** This works under the direction of resource selection algorithm (schedule advisor) to identify resource access costs. It interacts with trade servers and negotiates for access to resources at low costs.
- **Deployment Agent:** This is responsible for activating task execution on the selected resource as per the scheduler's instruction. It periodically updates the status of task execution to JCA.

Grid Middleware

The grid middleware offers services that help in coupling a grid user through resource broker or grid enable application and (remote) resources. It offers core services such as remote process management, co-allocation of resources, storage access, information (directory), security, authentication, and Quality of Service (QoS) such as resource reservation and trading. The Globus middleware offers a number of these services [17][19] that we use in our work:

- Resource allocation and process management (GRAM).
- Unicast and multicast communications services (Nexus)
- Authentication and related security services (GSI)
- Distributed access to structure and state information (MDS)
- Monitoring of health and status of system components (HBM)
- Remote access to data via sequential and parallel interfaces (GASS)
- Construction, caching, and location of executables (GEM)
- Advanced resource reservation (GARA)

The resource trading services are offered by our middleware infrastructure, GRACE (see next section):

- GRid Architecture for Computational Economy

Local Resource Manager

The local resource manager is responsible for managing and scheduling computations across local resources such as workstations and clusters. They are even responsible for offering access to storage devices, databases, and special scientific instruments such as a radio telescope. The example local resource managers include, cluster operating systems such as MOSIX [28] and queuing systems such as LSF and Condor.

6. GRid Architecture for Computational Economy (GRACE)

The GRACE infrastructure is a middleware component that can co-exist with grid middleware systems such as Globus. It offers services that help resource brokers in dynamically trading (cheap) resource to support computational economy. The components of GRACE infrastructure are:

- A Trade Manger (it is actually a GRACE client and a component of the resource broker).
- Trading Protocols and APIs.
- A Trade Server (it uses pricing algorithms defined by the resource owner and interacts with Resource Usage Accounting and Billing system).

Grid Trade Manager and Trade Server

The Trade Manager (TM) is a client that uses GRACE trading APIs to interact with trade servers and negotiates for access to resources at low cost. It works under the direction of resource selection algorithm (schedule advisor) to identify resource access costs.

The Trade Server (TS) is a resource owner agent that negotiates with resource users and sells access to resources. It aims to maximize the resource utility and profit its owner (earn as much money as possible). It uses pricing algorithms as defined by the resource owner that may be driven by the demand and supply. It also interacts with the accounting system for recording resource usage that bills the user. In effect, we are employing a "competitive" market approach to resource allocation, wherein the TM tries to minimize the cost of computation for resource users and the TS tries to maximize the profit for resource owners.

Grid Open Trading Protocols and APIs

The Trading Protocols define the rules and format for exchanging commands and messages between GRACE client (Trade Manager) and Trade Server. Figure 2 shows multilevel protocols or steps that both client and server need to follow while trading for the cost of resource access. The wire-level (low-level) details of these protocols are skipped, as they are obvious.

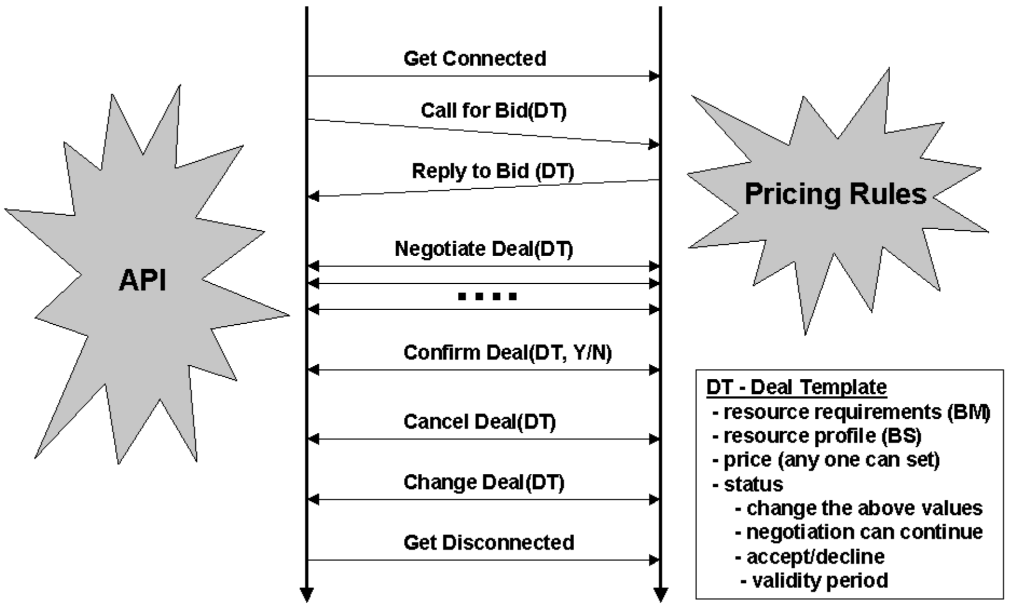


Figure 2: GRACE Open Trading Protocols.

The finite state machine representation of GRACE trading protocols is show in Figure 3. In our model, the Trade Manager (TM) contacts trade server with a request for a quote/bid. The TM specifies resource requirements in Deal Template (DT), which can be represented by a simple structure (record) with its fields corresponding to deal items or by a "Deal (Template) Specification Language" similar to the *ClassAds* mechanism employed by the Condor [10] system. The contents of DT include, CPU time units, expected usage duration, storage requirements, etc., along with its initial offer or leave it blank. The TM looks into DT and updates its contents with price etc., and sends back to TS. This negotiation between TM and TS continues until one of them says that its offer is final (no more negotiation). Then it is up to the other party to decide whether to accept or reject the deal. If accepted, then both works as per the agreement mentioned in the deal. The overhead introduced by the multilevel point-to-point protocol can be reduced when resource access prices are announced (like in the market) through GIS.

Grid Open Trading APIs

The GRACE infrastructure supports generic Application Programming Interfaces (APIs) that can be used by the grid tools and application programmers to develop software supporting the computational economy. The trading APIs are C-like functions (high level view of trading protocols) that GRACE clients can use to communicate with trading agents:

- `grid_trade_connect(resource_id, tid)`
- `grid_request_quote(tid, DT)`
- `grid_trade_negotiate (tid, DT)`
- `grid_trade_confirm(tid, DT)`
- `grid_trade_cancel(tid, DT)`
- `grid_trade_change( tid, DT)`
- `grid_trade_reconnect(tid, resource_id)`
- `grid_trade_disconnect(tid)`

where,

```
tid = Trade Identification code
DT = Deal Template
```

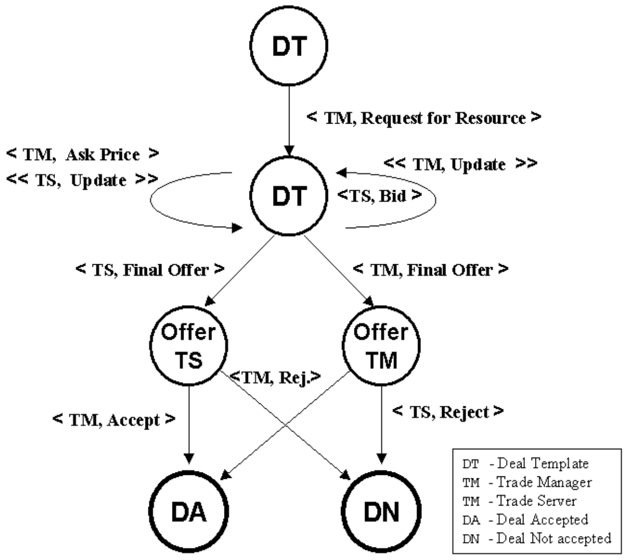


Figure 3: A Finite State Machine.

7. A new Nimrod/G Resource Broker

The Nimrod/G resource broker is a tailored global scheduler for running parametric applications on computational grid [1][6]. It is developed using Globus toolkit services and can be easily extended to operate with any other emerging grid middleware services. It uses MDS services for dynamic resource discovery and GRAM APIs to dispatch jobs over wide-area distributed grid resource. It allows scientists and engineers to model whole parametric experiments and transparently stage the data (using GASS) and program (using GEM) at remote sites, and run the program on each element of a data set on different machines and finally gather results from remote sites to the user site. The user need not worry about the way in which the complete experiment is set up, data or executable staging, or management. The user can also set the deadline by which the results are needed and the Nimrod/G broker tries to find the cheapest computational resources available in the grid and use them so that the user deadline is met and cost of computation is kept to a minimum. However, the grid resources are shared and their availability and load varies from time to time. When scheduler notices that it cannot meet the deadline with the current resource set, it tries to select the next cheapest resource and continues to do this until the completion of task farm application meets the (soft) deadline. We have performed a number of experiments using this approach on the GUSTO test-bed, and these are reported in [1]

The Nimrod/G (discussed in [1][6]) uses *static* cost model (stored in a file) for resource access cost trade-off with the deadline. In this paper we propose a new architecture for Nimrod/G resource broker (see Figure 4) to overcome the current limitation using GRACE middleware services (discussed earlier). It is possible to make a one-to-one mapping between the generic architecture of grid resource management system (shown in Figure 1) and the Nimrod/G architecture.

The key components of Nimrod/G system are Client/User Station, a Persistent Parametric/Task-farming Nimrod Engine, Scheduler, and Dispatcher (for detailed discussion of these components, see [1][6]). One of the key components of proposed Nimrod/G architecture is trading manager. The functionality of TM has already been discussed earlier sections. It will also explore the advance resource reservation during trading. We believe that with this new architecture, the Nimrod/G should be able to answer the user queries such as “I am willing to pay $$$, can you complete this job by deadline D?”. This ability means, users can trade-off the deadline against the cost and decide the manner in which computations are to be performed.

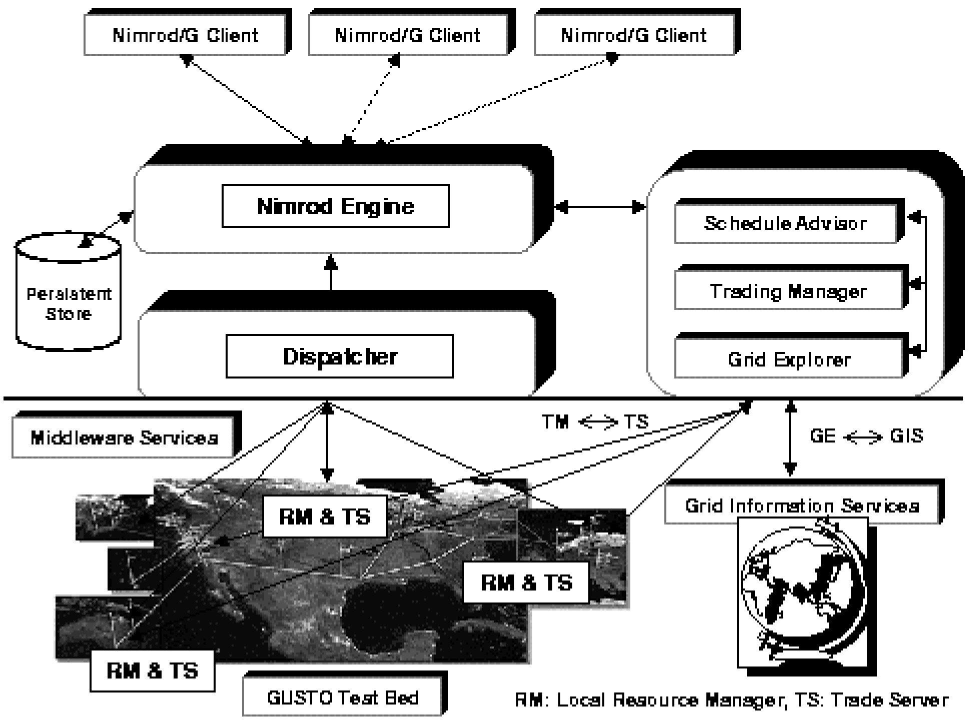


Figure 4: A new Nimrod/G Grid Resource Broker.

8. Conclusions and Future Work

We have discussed issues involved in the resource management architecture for computational grids. We identified a number of challenging problems including economy of computations that have driven the resource management architecture discussed in this paper. We discussed a new middle service infrastructure called GRid Architecture for Computational Economy (GRACE). Our future work focuses on the realization of various scheduling models driven by computational economy and incorporation of these into Nimrod/G resource broker. The scheduling algorithms that we would like to explore are based on reservation of resources in advance, and dynamic computational economy based on advertised costs, trading, and auction mechanisms. We plan to drive the scheduling work based on fuzzy logic and genetic algorithms.

We expect that economy driven approach to resource management will have impact on the success of the grid as much as the web had on the Internet! It enables us to build a truly scalable computational grid that follows user-centric approach in scheduling. In this, one can "sell" excess computational resource or "buy" when in need and thus commoditizing compute power!

Acknowledgements

We thank Sudharshan Vazhkudai, Spyros Lalis, Rob Gray, and David DiNucci for their comments on the paper. The support of IPRS, MGS, DSTC, DSSE, and IEEE Computer Society scholarships are acknowledged.

References

[1] Abramson, D., Giddy, J., and Kotler, L., *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?*, IPDPS'2000, Mexico, IEEE CS Press, USA, 2000.

[2] Almond J., Snelling D., *UNICORE: uniform access to supercomputing as an element of electronic commerce,* Future Generation Computer Systems 15(1999) 539-548, NH-Elsevier.

[3] Baker M., Buyya R., Laforenza D., *The Grid: International Efforts in Global Computing,* Intl. Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR'2000), Italy, 2000 (to appear).

[4] Brent C. and Culler, D., *Rexec: A decentralized, secure remote execution environment for clusters,* 4th Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing, France, 2000.

[5] Buyya, R. (ed.), *High Performance Cluster Computing: Architectures and Systems*, Volume 1 and 2, Prentice Hall PTR, NJ, USA, 1999.

[6] Buyya, R., Abramson, D., and Giddy, J., *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.

[7] Berman F. and Wolski R., *The AppLeS Project: A Status Report*, Proceedings of the Eight NEC Research Symposium, Germany, May 1997.

[8] Chapin S., Clement M., and Snell Q., *Strawman 1: A Grid Resource Management Architecture,* Grid Forum Scheduling Working Group, Nov. 1999.

[9] Casanova H. and Dongarra, J., *NetSolve: A Network Server for Solving Computational Science Problems*, Intl. Journal of Supercomputing Applications and High Performance Computing, Vol. 11, No. 3, 1997.

[10] Condor - http://www.cs.wisc.edu/condor/

[11] Czajkowski K., Foster I., Karonis N., Kesselman C., Martin S., Smith W., and Tuecke S., *A Resource Management Architecture for Metacomputing Systems,* IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[12] DiNucci D., *Abstract Owner (AO) Strawman*, Grid Forum Scheduling Working Group, Dec. 1999.

[13] DISCWorld - http://dhpc.adelaide.edu.au/

[14] Distributed.Net – http://www.distributed.net/

[15] Distributed ASCI Supercomputer (DAS) - http://www.cs.vu.nl/das/

[16] Dongarra J., *An Overview of Computational Grids and Survey of a Few Research Projects*, Symposium on Global Information Processing Technology, Japan, 1999.

[17] Foster I. and Kesselman C., *Globus: A Metacomputing Infrastructure Toolkit*, International Journal of Supercomputer Applications, 11(2): 115-128, 1997.

[18] Foster, I., and Kesselman, C. (editors), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[19] Globus Project - http://www.globus.org

[20] Globus Testbeds - http://www-fp.globus.org/testbeds/

[21] Grid Forum - http://www.gridforum.org/

[22] GF Scheduling Working Group, *General Meeting Notes,* 3rd Grid Forum Meeting, March'2000.

[23] Gridware, *Codine/GRD*, http://www.gridware.com

[24] JaWS - http://roadrunner.ics.forth.gr:8080/

[25] Legion - http://legion.virginia.edu/

[26] LSF Home Page - http://www.platform.com

[27] Mariposa-http://mariposa.cs.berkeley.edu:8000/mariposa/

[28] MOSIX - http://www.mosix.cs.huji.ac.il/

[29] NASA IPG – http://www.ipg.nasa.gov

[30] Schopf J., Nitzberg B., Chapin S., Clement M., and Snell Q., *Strawman2: A Grid Resource Management Architecture*, GF Scheduling Working Group, Dec. 1999.

[31] SETI@Home – http://setiathome.ssl.berkeley.edu/

[32] Ninf - http://ninf.etl.go.jp/

[33] Thigpen B. and Hacker T., *Distributed Accounting on the Grid*, The Grid Forum Working Drafts, 2000.