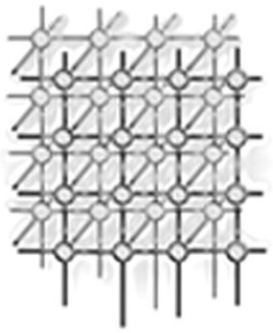
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2002; **14**:1507–1542 (DOI: 10.1002/cpe.690)

Economic models for resource management and scheduling in Grid computing



Rajkumar Buyya[1,*,†], David Abramson[2], Jonathan Giddy[2] and Heinz Stockinger[3]

[1] *Grid Computing and Distributed Systems (GRIDS) Lab., Department of Computer Science and Software Engineering, The University of Melbourne, 221 Bouverie St., Carlton, Melbourne, Australia*
[2] *CRC for Enterprise Distributed Systems Technology, School of Computer Science and Software Engineering, Monash University, Melbourne, Australia*
[3] *CMS Experiment, Computing Group, CERN, European Organization for Nuclear Research, CH-1211 Geneva 23, Switzerland*

SUMMARY

The accelerated development in peer-to-peer and Grid computing has positioned them as promising next-generation computing platforms. They enable the creation of *virtual enterprises* for sharing resources distributed across the world. However, resource management, application development and usage models in these environments is a complex undertaking. This is due to the geographic distribution of resources that are owned by different organizations or peers. The resource owners of each of these resources have different usage or access policies and cost models, and varying loads and availability. In order to address complex resource management issues, we have proposed a computational economy framework for resource allocation and for regulating supply and demand in Grid computing environments. This framework provides mechanisms for optimizing resource provider and consumer objective functions through trading and brokering services. In a real world market, there exist various economic models for setting the price of services based on *supply-and-demand* and their value to the user. They include commodity market, posted price, tender and auction models. In this paper, we discuss the use of these models for interaction between Grid components to decide resource service value, and the necessary infrastructure to realize each model. In addition to usual services offered by Grid computing systems, we need an infrastructure to support interaction protocols, allocation mechanisms, currency, secure banking and enforcement services. We briefly discuss existing technologies that provide some of these services and show their usage in developing the Nimrod-G grid resource broker. Furthermore, we demonstrate the effectiveness of some of the economic models in resource trading and scheduling using the Nimrod/G resource broker, with deadline and cost constrained scheduling for two different optimization strategies, on the World-Wide Grid testbed that has resources distributed across five continents. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: world-wide computing; grid economy; resource management; scheduling

*Correspondence to: Rajkumar Buyya, Grid Computing and Distributed Systems (GRIDS) Lab., Department of Computer Science and Software Engineering, The University of Melbourne, 221 Bouverie St., Carlton, Melbourne, Australia.
†E-mail: raj@cs.mu.oz.au

Received 30 July 2001
Revised 6 January 2002

1. INTRODUCTION

Computational Grids and peer-to-peer (P2P) computing systems are emerging as a new paradigm for solving large-scale problems in science, engineering and commerce [1,2]. They enable the creation of *virtual enterprises* (VEs) for *sharing* and *aggregation* of millions of resources (e.g. SETI@Home [3]) geographically distributed across organizations and administrative domains. They comprise heterogeneous resources (PCs, workstations, clusters and supercomputers), fabric management systems (single system image OS, queuing systems, etc.) and policies, and applications (scientific, engineering and commercial) with varied requirements (CPU, I/O, memory and/or network intensive). The *producers* (resource owners) and *consumers* (resource users) have different goals, objectives, strategies and supply-and-demand patterns. More importantly, both resources and end-users are geographically distributed with different time zones. In managing such complex environments, traditional approaches to resource management that attempt to optimize system-wide measure of performance cannot be employed. Traditional approaches use centralized policies that need complete state information and a common fabric management policy, or a decentralized consensus-based policy. Due to the complexity in constructing successful Grid environments, it is impossible to define an acceptable system-wide performance matrix and common fabric management policy [4]. (The concepts discussed in this paper apply to both P2P and Grid systems although we can argue about some of their technical, social and political differences. However, we use the term Grid for simplicity and brevity.)

In [2,5–7], we proposed and explored the usage of an economics-based paradigm for managing resource allocation in Grid computing environments. The economic approach provided a fair basis in successfully managing decentralization and heterogeneity that is present in human economies. Competitive economic models provide algorithms/policies and tools for resource sharing or allocation in Grid systems. The models can be based on bartering or prices. In the *bartering-based model*, all participants need to own resources and trade resources by exchanges (e.g., storage space for CPU time). In the *price-based model*, the resources have a price, based on the demand, supply, value and the wealth in the economic system.

We envision a future in which economically intelligent and economically motivated P2P and Grid-like software systems will play an important role in distributed service-oriented computing. The resource management systems need to provide mechanisms and tools that facilitate the realization of the goals of both resource owners and users. The resource consumers' need a *utility model*—to allow them to specify resource requirements and preference parameters—and *brokers* that provide strategies for choosing appropriate resources that meet user requirements. The resource owners need mechanisms for *price generation schemes* to increase system utilization and *protocols* that help them offer competitive services. For the market to be competitive and healthy, coordination mechanisms are required that help the market reach an equilibrium price—the price at which the supply of a service equals the quantity demanded.

Most of the related work in Grid computing dedicated to resource management and scheduling problems adopt a *conventional style* where a scheduling component decides which jobs are to be executed at which site based on certain cost functions (Legion [8], Condor [9], AppLeS [10], Netsolve [11], Punch [12]). Such cost functions are often driven by system-centric parameters that enhance system throughput and utilization rather than improving the utility of application processing. They treat resources as if they all cost the same price and the results of all applications have the same value even though this may not be the case in reality. The end-user does not want to pay the highest

price but wants to negotiate a particular price based on the demand, value, priority and available budget. Also, the results of different applications have different values at different times. In an economics approach, the scheduling decision is not done statically by a single scheduling entity but directed by the end-users' requirements. Whereas a conventional cost model often deals with software and hardware costs for running applications, the economic model primarily charges the end-user for services that they consume based on the value they derive from it. Pricing based on the demand of users and the supply of resources is the main driver in the competitive, economic market model. Thus, we stress that a user is in competition with other users and a resource owner with other resource owners.

The main contribution of this paper is to provide economic models, system architecture and policies for resource management in Grid environments. Currently, the user community and the technology are still rather new and not well accepted and established in commercial settings. However, we believe the Grid can become established in such settings by providing an incentive to both consumers and resource owners for being part of the Grid. Since the Grid uses the Internet as a carrier for providing remote services, it is well positioned to create a computational ecology, cooperative problem solving environment and means for sharing computational and data resources in a seamless manner. Up until now, the idea of using Grids for solving large computationally intensive applications has been more or less restricted to the scientific community. However, even if, in the scientific community, the pricing aspect seems to be of minor importance, funding agencies need to support the hardware and software infrastructure for Grids. Economic models help them manage and evaluate resource allocations to user communities. The system managers may impose quota limitations and value different resources with a different number of tokens [13]. In such environments, resource consumers certainly prefer to use economic-driven schedulers to effectively utilize their tokens by using lightly loaded cheaper resources.

2. PLAYERS IN THE GRID MARKETPLACE

The two key players driving the Grid marketplace are *Grid service providers* (GSPs) providing the traditional role of *producers* and *Grid resource brokers* (GRBs) representing *consumers*. The Grid computing environments provide the necessary infrastructure including security, information, transparent access to remote resources and information services that enable us to bring these two entities together [2]. Consumers interact with their own brokers for managing and scheduling their computations on the Grid. The GSPs make their resources Grid enabled by running software systems (such as Globus [14] or Legion [8]) along with *Grid trading services* to enable resource trading and execution of consumer requests directed through GRBs. The interaction between GRBs and GSPs during resource trading (service cost establishment) is mediated through a *Grid market directory* (GMD) (see Figures 1–5). They use various economic models or interaction protocols for deciding service access price. These models are discussed in Section 3. Our Grid Architecture for Computational Economy (GRACE), proposed in our earlier work [2], provides detailed information on the role played by GSPs, GRBs and GMDs and compares them to actual Grid components and implementations.

As in the conventional marketplace, the users' community (GRBs) represents the demand, whereas the resource owners' community (GSPs) represents the supply. We emphasize the user community and how they can influence the pricing of Grid resources via their brokers. Numerous economic models including microeconomic and macroeconomic principles have been proposed in the literature. Some of

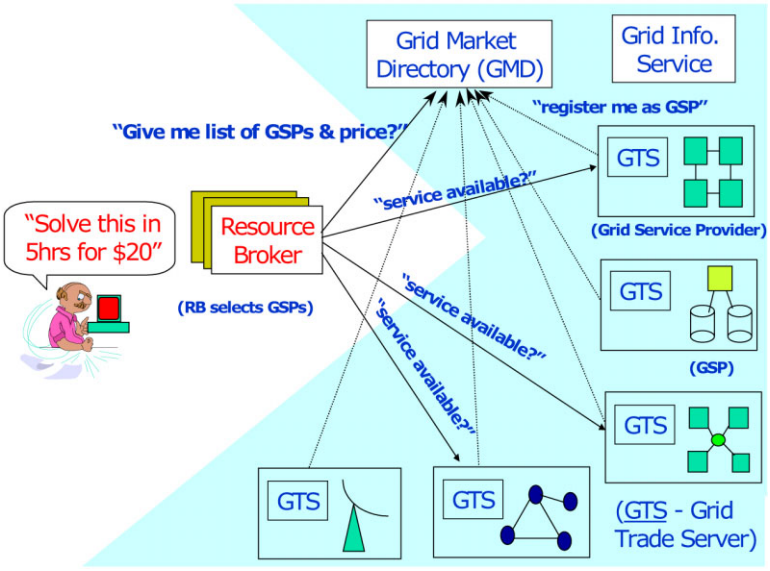


Figure 1. Interaction between GSPs and users in a commodity market Grid for resource trading.

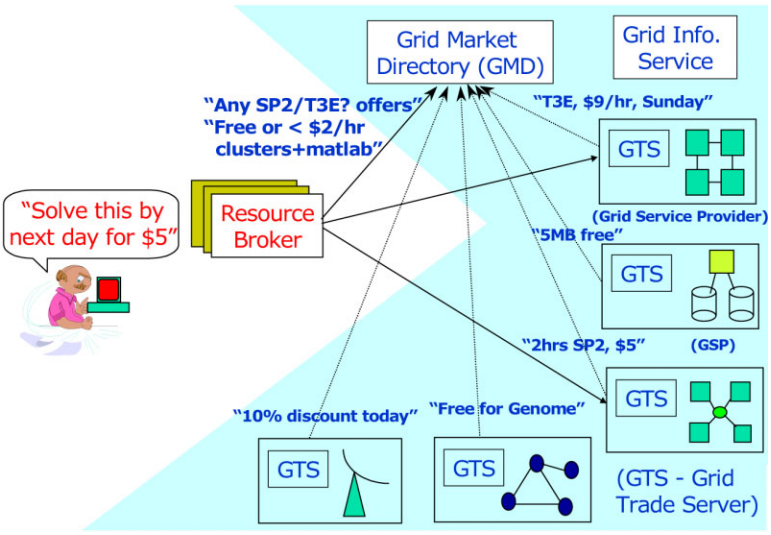


Figure 2. Posted price model and resource trading in a computational market environment.

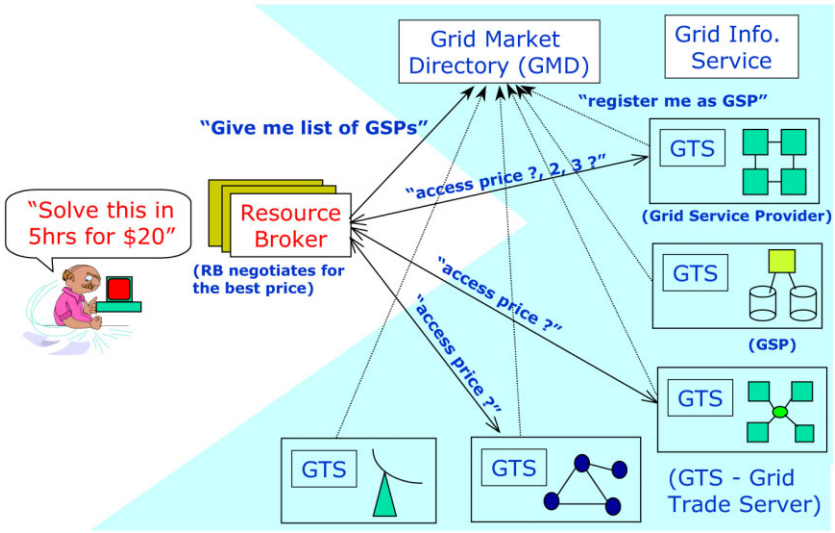


Figure 3. Brokers bargaining for lower access prices in their bid for minimizing computational cost.

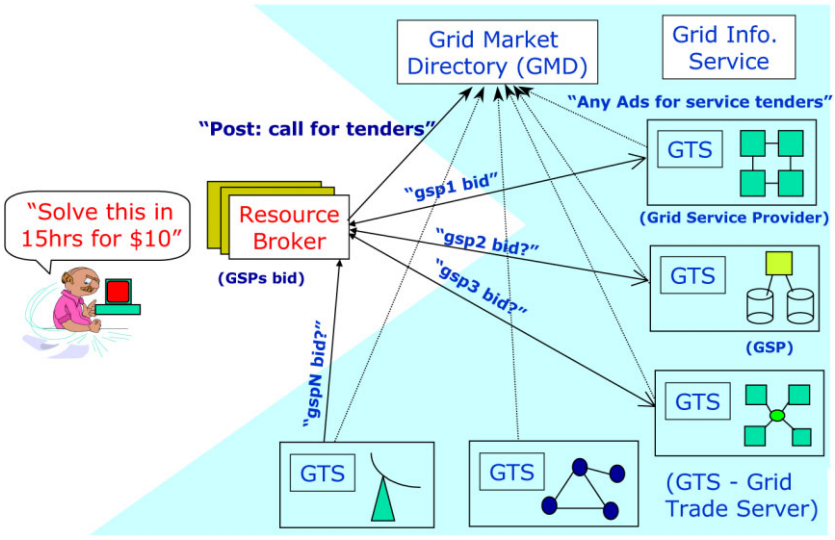


Figure 4. Tender/contract-net model for resource trading.

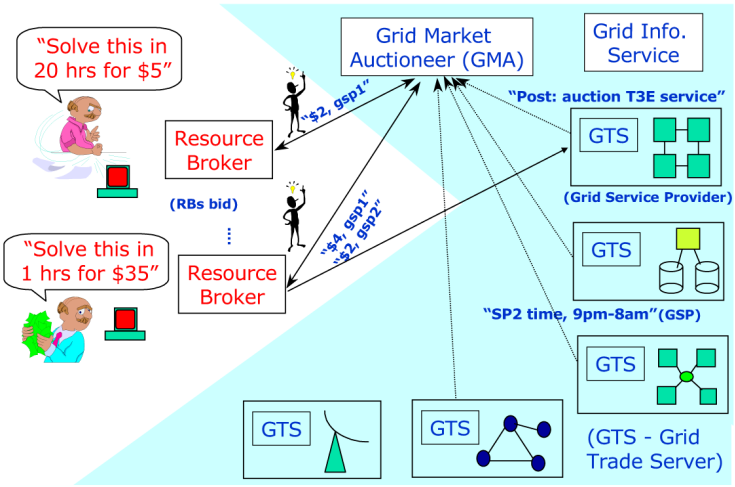


Figure 5. Auctions using an external auctioneer.

the commonly used economic models that can be employed for managing resources environment, include:

- the commodity market model;
- the posted price model;
- the bargaining model;
- the tendering/contract-net model;
- the auction model;
- the bid-based proportional resource sharing model;
- the community/coalition/bartering model;
- the monopoly and oligopoly.

Several research systems (see Table I) have attempted to apply the concept of computational economy models to database management, CPU cycles, storage and distributed computing. They include Mariposa [15], Mungi [16], Popcorn [17], JavaMarket [18], Enhanced MOSIX [19], JaWS [20], Xenoservers [21], D'Agents [22], Rexec/Anemone [23], Spawn [25], Mojo Nation [24] and Nimrod-G [1,5]. These systems have been targeted to manage single or multiple resources and they can be categorized as follows.

- *Single domain computing systems:* Enhanced MOSIX and Rexec/Anemone.
- *Agent-based systems:* Xenoservers and D'Agents.
- *Distributed database management system:* Mariposa.
- *Shared storage management system:* Mungi.
- *Web-based distributed systems:* Popcorn, JavaMarket, and JaWS.
- *Multi-domain distributed Grid system:* Nimrod-G.

Table I. Computational economy-based resource management systems.

System name	Economy model	Platform	Remarks
Mariposa [15] (UC Berkeley)	Bidding (tendering/contract-net). Pricing based on load and historical info	Distributed database	It supports budget-based query processing and storage management
Mungi [16] (University of New South Wales)	Commodity market (renting storage space that increases as available storage runs low, forcing users to release unneeded storage)	Storage servers	It supports storage objects based on bank accounts from which rent is collected for the storage occupied by objects
Nimrod-G [1,5] (Monash University)	It supports economy models such as commodity market, spot market and contract-net for price establishment	A Grid of distributed computers (PCs, workstations and clusters)	It supports deadline and budget constrained scheduling algorithms for executing task-farming applications on distributed resources depending on their cost, power and availability and users quality of service requirements
Popcorn [17] (Hebrew University)	Auction (highest bidder gets access to resource and it transfers credits from buyer to the seller account)	Web browsers (*Popcorn parallel code* runs within a browser of CPU cycles seller)	Popcorn API-based parallel applications need to specify a budget for processing each of its modules
JavaMarket [18] (Johns Hopkins University)	Quality of service (QoS) based computational market (the resource owner receives $f(j,t)$ award for completing f in time t)	Web browsers (JavaMarket runs *standard Java Applets* within a browser)	One can sell CPU cycles by pointing Java-enabled browser to portal and allowing execution of Applets
Enhanced MOSIX [19] (Hebrew University, Israel)	Commodity market (resource cost of each node is known)	Clusters of computers (Linux PCs)	It supports process migration such that overall cost of job execution is kept low
JaWS [20] (University of Crete)	Bidding (tendering)	Web browsers	It is similar to Popcorn
Xenoservers [21] (University of Cambridge)	Bidding (proportional resource sharing)	Single computer	Accounted execution of untrusted code
D'Agents [22] (Dartmouth College)	Bidding (proportional resource sharing)	Single computer or Mobile Agents	Agents bid function is proportional to benefit
Rexec/Anemone [23] (UC Berkeley)	Bidding/auction (for proportional resource sharing)	Clusters (A market-based cluster batch queue system)	Users assign utility value to their application and system allocates resources proportionally
Mojo Nation [24] (Autonomous Zone Industries, CA)	A credit-based partnership and/or bartering model (contributors earn credits by sharing storage and spend them when required)	Network storage	It is a content-sharing community network It combines marketplace and bartering approach for file/resource sharing
Spawn [25] (Xerox PARC)	Second-price auction (uses sponsorship model for funding money to each task depending on some requirements)	Network on workstations. Each workstation executes a single task per time slice	It supports execution of a concurrent program expressed in the form of a hierarchy of processes that expand and shrink in size depending on the resource cost
Supercomputing Centre [13] (University of Manchester)	Commodity market and priority-based model (they charge for CPU, memory, storage and human support services)	MPPs, Crays and Clusters, and storage servers	Any application can use this service and QoS is proportional to user priority and scheduling mechanisms

Many of the resource management systems presented in Table I support a single model for resource trading, provide their own programming model and are implemented as monolithic systems. To overcome these limitations, the modern Grid computing systems use a layered architecture. Typically, in a Grid marketplace, the resource owners and users can use any one or more of these models or even combinations of them in meeting their objectives [2]. Both have their own expectations and strategies for being part of the Grid. The resource consumers adopt the strategy of solving their problems at low cost within a required time frame. The resource providers adopt the strategy of obtaining best possible return on their investment while trying to maximize their resource utilization by offering a competitive service access cost in order to attract consumers. The resource consumers can choose providers that best meet their requirements. The design and architecture for the development of Grid systems using these economic models is discussed in the next section.

Both GRBs and GSPs can initiate resource trading and participate in the interaction depending on their requirements and objectives. GRBs may invite bids from a number of GSPs and select those that offer the lowest service costs and meet their deadline and budget requirements. Alternatively, GSPs may invite bids in an auction and offer services to the highest bidder as long as its objectives are met. Both GSPs and GRBs have their own utility functions that must be satisfied and maximized. The GRBs perform a cost–benefit analysis depending on the deadline (by which the results are required) and budget available (the amount of money the user is willing to invest for solving the problem). The resource owners decide their pricing based on various factors [2]. They may charge different prices for different users for the same service or it can vary depending on the specific user demands. Resources may have different prices based on environmental influences such as the availability of larger core memory and better communication bandwidth with an outside world.

Grid brokers (note that in a Grid environment each user has his own broker as his agent) may have different goals (e.g., different deadlines and budgets), and each broker tries to maximize its own good without concern for the global good. However, given adequate information, independent rational self-interest achieves the optimal resource allocation, both for the individual and for the society. This needs to be taken into consideration when building an automated negotiation infrastructure. In a *cooperative distributed computing or problem-solving environment* (like cluster computers), the system designers impose an *interaction protocol* (possible actions to take at different points) and a *strategy* (a mapping from one state to another and a way to use the protocol). This model aims for global efficiency as nodes cooperate towards a common goal. On the other hand, in Grid systems, brokers and GSPs are provided with an interaction protocol, but they choose their own private strategy (like in multi-agent systems), which cannot be imposed from outside. Therefore, the negotiation protocols need to be designed assuming a *non-cooperative, strategic* perspective. In this case, the main concern is what social outcomes follow given a protocol, which *guarantees that each broker/GSP's desired local strategy is best for that broker/GSP and hence the broker/GSP will use it*.

The various criteria used for judging the effectiveness of a market model are [26]:

- social welfare (global good of all);
- Pareto efficiency (global perspective);
- individual rationality (better off by participating in negotiation);
- stability (mechanisms that cannot be manipulated, i.e. behave in the desired manner);
- computational efficiency (protocols should not consume too much computation time);
- distribution and communication efficiency (communication overhead to capture a desirable global solution).

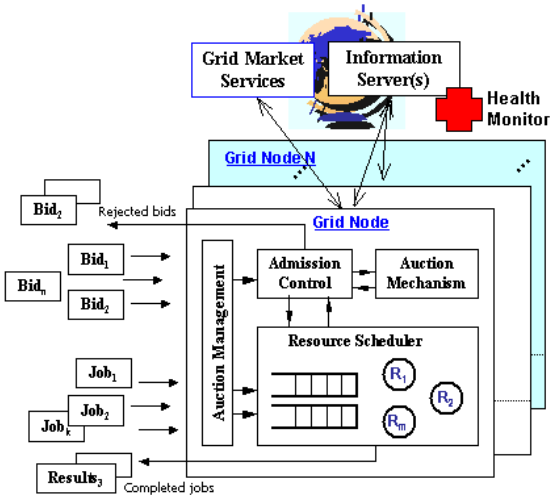


Figure 6. An auction using their own auctioneer.

3. ECONOMIC MODELS IN A GRID CONTEXT

In the previous section we identified a few popular models that are used in human economies. In this section we discuss the use of these economic models and propose an architecture for realizing them. The discussion on realizing negotiation protocols based on different economic models is kept as generic as possible. This ensures that our proposed architecture is free from any specific implementation and that it provides a general framework for any other Grid middleware and tool developers. Particular emphasis will be placed on the framework and heuristics that GRBs can employ for establishing their service price depending on their customers' requirements. The service providers publish their services through the GMD. They use the declarative language of the Grid trading services to define the cost specifications and their objectives, such as access price for various users for different times and durations along with the possibilities of offering discounts to attract users during off-peak hours. The Grid trading server (GTS) can employ different economic models in providing services. The simplest would be a commodity model wherein the resource owners define pricing strategies including those driven by the demand and resource availability. The GTS can act as auctioneer if an auction-based model is used in deciding the service access price or an external auctioneer service can be used (see Figure 6).

For each of the economic models, firstly, the economic model theory, its parameters and influences are discussed, and then a possible solution is given for a current Grid environment and how they can be mapped to existing Grid tools and architectures or what needs to be extended. In the classical economic theory there are different models for specific environmental situations and computing applications.

Since the end-user interaction is the main interest of this paper, we point out possible interactions with the broker.

3.1. Commodity market (flat or supply-and-demand driven pricing) model

In the commodity market model, resource owners specify their service price and charge users according to the amount of resource they consume. The pricing policy can be derived from various parameters, and can be *flat* or *variable* depending on the resource *supply and demand*. In general, services are priced in such a way that supply-and-demand equilibrium is maintained. In the *flat price model*, once pricing is fixed for a certain period, it remains the same irrespective of service quality. It is not significantly influenced by the demand, whereas in a *supply-and-demand* model prices change very often based on supply-and-demand changes. In principle, when the demand increases or supply decreases, prices are increased until there exists an equilibrium between supply and demand. Pricing schemes in a commodity market model can be based on:

- flat fee;
- usage duration (time);
- subscription;
- demand and supply [27].

The resource owners publish their prices through yellow pages like the GMD service (see Figure 1). This is accomplished by defining the price specification that the GTS can use for publishing the service access price in the market directory. A simple price specification may contain the following parameters.

```
. . .
consumer_id                    // this can be same Grid-ID
peak_time_price                // 9am-6pm: office hours on working days
lunch_time_price               // (12.30-2pm)
offpeak_time_price             // (6pm-9am),
discount_when_lightly_loaded   // if load is less than 50% at any time
raise_price_high_demand        // % raise price if average load is above 50%
price_holiday_time             // during holidays and week ends!
. . .
```

Traditionally, computational services are priced based on their production cost and desired profit margin. However, the consumers' perception of value is based on parameters such as supply and demand for resources, priority and service quality requirements. Therefore, the resource value in a Grid economy needs to be defined as a function of many parameters:

$$\text{Resource value} = \text{Function}(\text{resource strength, cost of physical resources, service overhead, demand, value perceived by the user, preferences}).$$

The last three parameters are difficult to capture from consumers unless they see any benefit in disclosing them and they may vary from time to time, from one application to another. However, there are consumers who prefer regular access to resources during a particular period of the day. For example, those involved in making regular decisions on the supply chain management of goods (i.e. shipping

from the inventory to department stores) prefer calendar-based guaranteed access and stable but competitive pricing of resources; they do not require spot-market-based access to services [28]. In this case demand and preferences are clear, the pricing policy can be easily negotiated in advance in a competitive and reasonable manner and the resource quality of services can be guaranteed through reservation during the required period as agreed in advance.

Consumers can be charged for access to various resources including CPU cycles, storage, software and the network. The users compose their application using higher-level Grid programming languages. For example, in our Nimrod problem-solving environment we provide a declarative programming language for composing parameter sweep applications and for defining application and user requirements such as deadlines and budgets [1]. The resource broker (working for the user) can carry out the following steps for executing applications:

1. the broker identifies service providers;
2. it identifies suitable resources and establishes their prices (by interacting with the GMD and GTS);
3. it selects resources that meet its utility function and objectives (lower cost and deadline requirements met); it uses heuristics and/or historical knowledge while choosing resources and mapping jobs to them;
4. it uses resource services for job processing and issues payments as agreed;

As we are focusing on a generic framework, implementation specific details for releasing the above steps are not presented. For example, implementation specific details of our Nimrod/G resource broker [1,5,7] vary from other related systems.

3.2. Posted price model

The posted price model is similar to the commodity market model, except that it advertises special offers (see Figure 2) in order to attract (new) consumers to establish market share or motivate users to consider using cheaper slots. In this case, brokers need not negotiate directly with GSPs for price, but use posted prices as they are generally cheaper compared to regular prices. The posted price offers will have usage conditions, but they might be attractive for some users. For example, during holiday periods, demand for resources is likely to be limited and GSPs can post tempting offers or prices aiming at attracting users to increase resource utilization. The activities that are specifically related to the posted price model in addition to those related to commodity market model are:

1. GSPs post their special offers and associated conditions etc. in the GMD;
2. a broker looks at the GMD to identify if any of these posted services are available and fit its requirements;
3. a broker enquires (GSP) about the availability of posted services;
4. other steps are similar to those pointed out in the commodity market model.

3.3. Bargaining model

In the previous models, the brokers pay access prices, which are fixed by GSPs. In the bargaining model, resource brokers bargain with GSPs for lower access prices and higher usage durations.

Both brokers and GSPs have their own objective functions and they negotiate with each other as long as their objectives are met. The brokers might start with a very low price and GSPs with a higher price. They both negotiate until they reach a mutually agreeable price (see Figure 3) or one of them is not willing to negotiate any further. This negotiation is guided by user requirements (e.g., a deadline is too relaxed) and brokers can take risks and negotiate for cheaper prices as much as possible, and they can discard expensive machines. This might lead to lower utilization of resources, so GSPs might be willing to reduce the price instead of wasting resource cycles. Brokers and GSPs generally employ this model when market supply and demand and service prices are not clearly established. The users can negotiate for a lower price with the promise of some kind of favour or the promise of using a GSP's services even in the future.

3.4. Tender/contract-net model

Tender/contract-net model is one of the most widely used models for service negotiation in a distributed problem-solving environment [29]. It is modeled on the contracting mechanism used by businesses to govern the exchange of goods and services. It helps in finding an appropriate service provider to work on a given task. Figure 4 illustrates the interaction between brokers and GSPs in their bid to meet their objectives. A user/resource broker asking for a task to be solved is called the *manager* and the resource that might be able to solve the task is called the potential *contractor*. From a manager's perspective, the process is:

1. the consumer (broker) announces its requirements (using a deal template) and invites bids from GSPs;
2. interested GSPs evaluate the announcement and respond by submitting their bids;
3. the broker evaluates and awards the contract to the most appropriate GSP(s);
4. the broker and GSP communicate privately and use the resource (R).

The contents of the deal template used for work announcements include the addressee (user), the eligibility requirements specifications (for instance, Linux, x86arch and 128 MB memory), the task/service abstraction, an optional price that the user is willing to invest, the bid specification (what should the offer contain) and the expiration time (the deadline for receiving bids).

From a contractor's/GSP perspective, the process is:

1. receive tender announcements/advertisements (say in the GMD);
2. evaluate the service capability;
3. respond with a bid;
4. deliver service if a bid is accepted;
5. report results and bill the broker/user as per the usage and agreed bid.

The advantage of this model is that if the selected GSP is unable to deliver a satisfactory service, the brokers can seek the services of other GSPs. This protocol has certain disadvantages. A task might be awarded to a less capable GSP if a more capable GSP is busy at the award time. Another limitation is that the GRB manager has no obligation to inform potential contractors that an award has already been made. Sometimes a manager may not receive bids for several reasons: (a) all potential GSPs are busy with other tasks; (b) a potential GSP is idle but ranks the proposed tender/task below the other tasks under consideration; (c) no GSPs, even if idle, are capable of offering a service (e.g., the resource

is Windows NT-based, but the user wants Linux). To handle such cases, a GRB can request quick response bids to which GSPs respond with messages such as *eligible, busy, ineligible* or *not interested*. This helps the GRB in making changes to its work plan. For example, the user can change the deadline or budget to wait for new GSPs or to attract existing GSPs to submit bids.

The tender model allows directed contracts to be issued without negotiation. The selected GSP responds with an *acceptance* or *refusal* of an award. This capability can simplify the protocol and improve the efficiency of certain services.

3.5. Auction model

The auction model supports one-to-many negotiation, between a service provider (seller) and many consumers (buyers), and reduces negotiation to a single value (i.e. price). The auctioneer sets the rules of auction, acceptable for the consumers and the providers. Auctions basically use market forces to negotiate a clearing price for the service.

In the real world, auctions are used extensively, particularly for selling goods/items within a set duration. The three key players involved in auctions are: resource owners, auctioneers (mediators) and buyers (see Figure 5). Many e-commerce portals such as Amazon.com and eBay.com are serving as mediators (auctioneers). Both buyers' and sellers' roles can also be automated. In a Grid environment, providers can use an auction protocol for deciding service value/price (see Figure 6). The steps involved in the auction process are:

1. a GSP announces their services and invites bids;
2. brokers offer their bids (and they can see what other consumers offer if they like—depending on how open/closed);
3. step 2 goes on until no one is willing to bid a higher price or the auctioneer stops if the minimum price line is not met;
4. the GSP offers the service to the one who wins;
5. the consumer uses the resource.

Auctions can be conducted as open or closed depending on whether back-and-forth offers and counter offers are allowed. The consumer may update the bid and the provider may update the offered sale price. Depending on these parameters, auctions can be classified into five types:

- English auction (first-price open cry);
- first-price sealed-bid auction;
- Vickrey (second-price sealed-bid) auction [30];
- Dutch auction;
- double auction (continuous).

English auction (first-price open cry). All bidders are free to increase their bids exceeding other offers. When none of the bidders are willing to raise the price anymore, the auction ends, and the highest bidder wins the item at the price of his bid. In this model, the key issue is how GRBs decide how much to bid. A GRB has a private value (as defined by the user) and can have a strategy for a series of bids as a function of its private value and prior estimation of other bidder's valuations, and the past bids of others. The GRB decides the private value depending on the user-defined requirements (mainly

deadline and budget that he is willing to invest in the solution of the problem). In the case of private-value English auctions, a GRB's dominant strategy is to always bid a small amount 'higher' than the current highest bid, and stop when its private-value price is reached. In correlated value auctions, the policies are different and they allow the auctioneer to increase the price at a constant rate or at the rate he wishes. Those not interested in bidding anymore can openly declare so (open-exit) without the possibility of re-entry. This information helps other bidders and gives them a chance to adjust their valuation.

First-price sealed-bid auction. Each bidder submits one bid without knowing the others' bids. The highest bidder wins the item at the price of his bid. In this case a broker bid strategy is a function of the private value and the prior beliefs of other bidders' valuations. The best strategy is to bid less than the true valuation and one might still win the bid, but it all depends on what the others bid.

Vickrey (second-price sealed-bid) auction. Each bidder submits one bid without knowing the others' bids. The highest bidder wins the item at the price of the second highest bidder [30]. The implementation architecture and strategies are similar to the tender/contract-net model discussed earlier.

Dutch auction. The auctioneer starts with a high bid/price and continuously lowers the price until one of the bidders takes the item at the current price. It is similar to a first-price sealed-bid auction because in both cases the bid matters only if it is the highest, and no relevant information is revealed during the auction process. From the broker's bidding strategic point of view, a Dutch auction is similar to an English auction (first-price sealed-bid auction). The key difference between them is that in an English auction the bids start with a low opening and increase progressively until demand falls, whereas in a Dutch auction the bids start with a high opening price and decrease progressively until demand rises to match supply.

The interaction protocols for a Dutch auction are as follows: the auction attempts to find the market price for some goods by starting a price much higher than the expected market value, then progressively reducing the price until one of the buyers accepts the price. The rate of reduction in price is up to the auctioneer and they have a reserve price below which not to go. If the auction reduces the price to the reserve price with no buyers, the auction terminates. In terms of real time, the Dutch auction is much more efficient as the auctioneer can decrease the price at a strategic rate and the first higher bidder wins. In an Internet wide auction, it is appealing in terms of automating the process wherein all parties can define their strategies for agents that can participate in multiple auctions to optimize their objective functions.

Double auction. This is one of the most common exchange institutions in the marketplace whose roots go back to ancient Egypt and Mesopotamia [31]. In fact, it is the primary economic model for trading of equities, commodities and derivatives in stock markets (e.g., NASDAQ). In the double auction model, buy orders (*bids*) and sell orders (*asks*) may be submitted at anytime during the trading period. If at any time there are open bids and asks that match or are compatible in terms of price and requirements (e.g., quantity of goods or shares), a trade is executed immediately. In this auction orders are ranked highest to lowest to generate demand and supply profiles. From the profiles, the maximum quantity exchanged can be determined by matching *asks* (starting with lowest price and moving up) with demand *bids* (starting with highest price and moving down). Researchers have developed software-based agents mechanisms to automate the double auction for stock trading with or without human interaction [32].

The double auction model has high potential for Grid computing. The brokers can easily be enabled to issue *bids* depending on the budget, deadline, job complexity, scheduling strategy and resource characteristics requirements, and GSPs can issue *asks* depending on current load and perceived demand, and price constraints. Both orders can be submitted to GMD agents that provide continuous clearance or matching services. Since bids are cleared continuously, both GRBs and GSPs can make instant decisions with less computational overhead and complexity.

All the above auctions differ in terms of whether they are performed as open or closed auctions and the offer price for the highest bidder. In open auctions, bidding agents can know the bid value of other agents and will have an opportunity to offer competitive bids. In closed auctions, the participants' bids are not disclosed to others. Auctions can suffer from collusion (if bidders coordinate their bid prices so that the bids stay artificially low), deceptive auctioneers in the case of a Vickrey auction (the auctioneer may overstate the second highest bid to the highest bidder unless that bidder can vary it), deceptive bidders, counter speculation, etc.

3.6. Bid-based proportional resource sharing model

Market-based proportional resource sharing systems are quite popular in cooperative problem-solving environments such as clusters (in a single administrative domain). In this model, the percentage of resource share allocated to the user application is proportional to the bid value in comparison to other users' bids. The users are allocated credits or tokens, which they can use to have access to resources. The value of each credit depends on the resource demand and the value that other users place on the resource at the time of usage. For example, consider two users wishing to access a resource with similar requirements, but the first user is willing to spend 2 tokens and the second user is willing to spend 4 tokens. In this case, the first user gets one-third of the resource share whereas the second user gets two-thirds of the resource share, which is proportional to the value that both users place on the resource for executing their applications.

This strategy is a good way of managing a large shared resource in an organization or resource owned by multiple individuals (like multiple departments in a university) can have a credit allocation mechanism depending on the investment they made. They can specify how much credit they are willing to offer for running their applications on the resource. For example, a user might specify low credits for non-interactive batch jobs and high credits for interactive jobs with high response times. GSPs can employ this model for offering a QoS for higher price paying customers in a shared resource environment (as shown in Figure 7). Example systems such as Rexec/Anemone, Xenoservers and D'Agents CPU market employ a proportional resource sharing model in managing resource allocations [2].

3.7. Community/coalition/bartering/share holders model

A community of individuals shares each other's resources to create a cooperative computing environment. Those who are contributing their resources to a common pool can get access to that pool. A sophisticated model can also be employed here for deciding how much resources share contributors can get. It can involve credits that one can earn by sharing a resource, which can then be used when needed. A system like Mojonation.net employs this model for storage sharing. This model works when those participating in the Grid have to be both service providers and consumers.

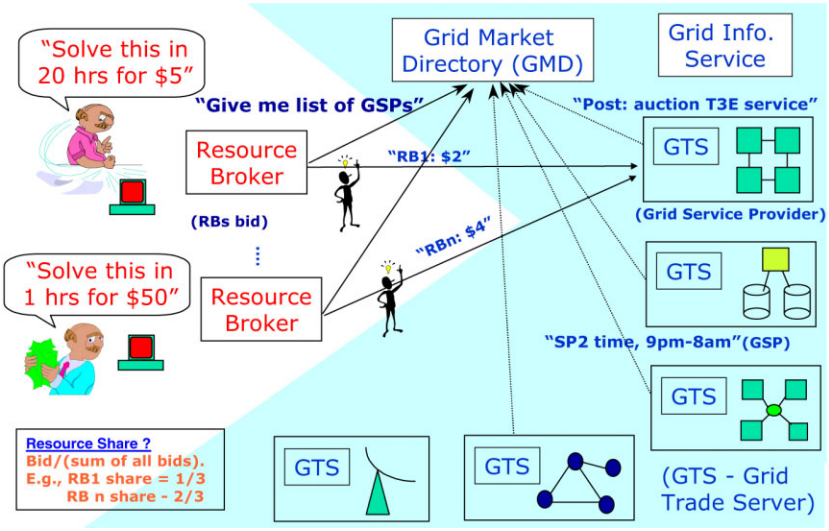


Figure 7. Market-based proportional resource sharing.

3.8. Monopoly/oligopoly

In the previously mentioned models we have assumed a competitive market where several GSPs and brokers/consumers determine the market price. However, there exist cases where a single GSP dominates the market and is therefore the single provider of a particular service. In economic theory this model is known as a monopoly. Users cannot influence the prices of services and have to choose the service at the price given by the single GSP who monopolizes the Grid marketplace. As regards the technical realization of this model, the single site puts the prices into the GMD or information services and brokers consult it without any possibility of negotiating prices.

The competitive markets are one extreme and monopolies are the other extreme. In most of the cases, the market situation is an *oligopoly*, which is in between these two extreme cases: a small number of GSPs dominate the market and set the prices.

3.9. Other influences on market prices

There are other influences on price setting strategies in competitive, international markets. Supply and demand is the most common one but one also has to take into account national borders and different pricing policies within different countries, such as taxation, consumer price index, inflation, etc. These factors are not dealt with in this paper, however implementations may need to consider them. There are also micro- and macro-economic factors that play an important role. One can also neglect them and build a price model on which all the Grid consumers have to agree. So this would correspond

to an international market with special rules. Then, a model has to be formed for price changes. What is the factor for that change? Is there a monopoly that can decide what to do? Is the market transparent with optimally adapted prices? These are some of the main questions that need to be answered by GSPs when they decide their prices in an international market. A broker may consult the Grid information service to find out where the price for a particular service is minimal. For instance, one country might impose special taxes on a service whereas another country does not.

There are occasions where resources are not valued as per the actual cost of resources and overheads involved in offering services. When new players enter the market, in order to attract customers from the existing GSPs they are likely to offer access at minimal price by undervaluing resources. This leads to price wars as GSPs are caught in a price cutting round to compete with each other. Measures such as the intervention of *price regulation authorities* can be in place to prevent the market from collapsing or the market can be left to consolidate naturally.

4. ECONOMY IN A DATA GRID ENVIRONMENT

In computational Grid environments, large computational tasks that do not use very large amounts of data are solved. In data Grid environments [33], large amounts of data are distributed and replicated to several sites all around the globe. Here, efficient access to the data is more important than scheduling computational tasks. When accessing large data stores, the cost for accessing data is important [34]. Can a single user afford to access a third of all data in a petabyte data store? Certain restrictions and cost functions need to be imposed in order to obtain a good throughput for data access and to provide a fair response time for multiple users. An economic model can help achieve local or global efficiency. Paying higher prices can result in accessing a larger amount of data. How these prices can be mapped to the requirements of scientific users is still an open research issue. However, it is clear that some optimizations and restrictions for data access are required.

4.1. A case for economy in a scientific data grid environment

In this subsection we discuss the possible use of economy in a scientific data Grid environment, in particular in the DataGrid project [33]. We claim that scheduling based on economic models is different in a conventional 'business environment' than in the scientific community. Whereas in the first a Grid user pays explicitly a certain amount of money in order to get a service (e.g., pay €100 for running application x in 10 minutes somewhere in the Grid), in the scientific community an explicit payment does not seem to be useful.

As regards the DataGrid project, several possible applications can be identified that require scheduling. Here, we concentrate only on the scheduling of user requests to (replicated) data. Let us define the problem. We assume several concurrent users (of the order of 100), replicated data stores with several terabytes, up to petabytes of data each and a finite throughput of data servers at a single site. A site can have several data servers and a hierarchical disk pool with tapes, but each single site will have a restriction on the maximum amount of data it can serve at a time. Thus, an optimization problem occurs that tries to optimize the throughput for a large user community. For instance, a single user should not be able to request a terabyte of data per day and consequently use all the data

server resources. A fair scheduling concept is required that allows multiple, concurrent user requests. The problem can be identified as a high throughput problem. A scheduler at a single site has to take care of a fair scheduling concept. This compares to the conventional market model where a single user can increase his/her own throughput by offering to pay higher prices for a service. In the data Grid environment a single user is interested in analysing data and does not want to pay money explicitly for the physics analysis job. In other words, data analysis should not be restricted by resource prices since the main focus of physics analysis is to find some new information in large amounts of data rather than making money by selling data resources. However, this needs to be regulated to provide fair and improved quality of service to all users.

Data access in a data Grid environment needs to be measured, regulated and it has to be translated into costs, but it will not be expressed in currency units. One possible mapping of access requests to costs is to define the maximum data throughput of a site in the data Grid. Based on this, a maximum number of tokens can be distributed to the users. For instance, a distributed data server at a site is able to serve 10 TB of data per day and thus 10 000 000 tokens are available and distributed to possible users. By default, a single user may only access as much data as he has tokens. This gives other users a chance to access data. However, the amount of data that they access for a given token needs to be based on parameters such as demand, system load, QoS requested, etc. This helps in better managing resources and improving QoS offered by the system. The users can tradeoff between QoS and tokens.

The local site scheduler has to take care of admission control. Now, a single user might want to access more data than the available tokens. This requires re-distribution and negotiation of tokens. The negotiation step can then be done with the various economic models discussed earlier. Each user gets a certain amount of budget (expressed in tokens) and can be expressed in terms of pre-allocation priorities. For instance, users A and B are allocated X and Y numbers of tokens. Tokens can be renewed each day based on the current demand and the data server capabilities for serving data. The tokens can be converted into money when we compare it to a commercial environment (like 10 tokens for $1). The data Grid system can have different pricing policies for data access at different times to encourage the users to organize and plan themselves depending on their priorities. For example, charging users 10 tokens per MB of data access during the peak time and 6 tokens per MB of data access during the off-peak time can help users to manage and prioritize their work flow effectively. Accessing replicas from different sites may also result in having different prices for different replicas. Such policies lead to an economy in data grid environments. Consequently, tokens are assigned to users depending on the value/importance of users and their work. We conclude that in a scientific data Grid environment economic models can be applied, but a scheduling instance is still required that controls the overall throughput for a large user community.

4.2. Data economy

In [34] a cost model for distributed and replicated data over a wide-area network is presented. Cost factors for the model are the network, data server and application specific costs. Furthermore, the problem of job execution is discussed under the viewpoint of sending the job to the required data (code mobility) or sending data to a local site and executing the job locally (data mobility). In the economic model, the main focus is on executing a job at any site as long as the cost for job execution

is minimal. Based on the previous case study we summarize this topic by introducing the term *data economy* that applies to several fields where data is distributed over Grids. The domains include:

- content (like music, books, newspapers) sales using techniques such as micro-payments, aggregation, subscription and subsidy;
- a community or bartering model for content sharing;
- data and replica sites access in data Grid environments.

5. Nimrod-G: A COMPUTATIONAL ECONOMY DRIVEN GRID RESOURCE BROKER

In this section, we briefly discuss the Nimrod-G resource broker as an example of a Grid system that uses a computational economy-driven architecture for managing resources and scheduling a task farming application on a large-scale distributed Grid resource. The Nimrod-G toolkit and resource broker is developed by leveraging services provided by Grid middleware systems such as Globus, Legion, Condor and the GRACE trading mechanisms. These middleware systems provide a set of low-level protocols for secure and uniform access to remote resources; and services for accessing resource information and storage management. The modular and layered architecture of Nimrod-G is shown in Figure 8. A detailed discussion on the Nimrod system architecture and implementation [1,5], scheduling algorithms [7] and its ability to execute real-world applications, such as ionization chamber calibration [1] and drug design [35], on the Grid can be found elsewhere.

The Nimrod-G toolkit provides a simple declarative parametric modelling language for expressing parametric experiments. It uses novel resource management and scheduling algorithms based on economic principles. Specifically, it supports user-defined deadline and budget constraints for schedule optimizations and manages the supply and demand of resources in the Grid using a set of resource trading services provided by GRACE [6]. The GridBank (G-Bank) has been proposed as an infrastructure for managing accounts of resource owners and users along with handling electronic payments. A case for realizing an economy Grid for service-oriented computing has been presented in our earlier work [2].

The Nimrod-G resource broker is responsible for determining the specific requirements that an experiment places on the Grid and performing resource discovery, resource trading, scheduling, the deployment of jobs on Grid resources, the starting and managing of job execution and the gathering of results back to the home node. The four key components of the Nimrod-G resource broker are: the *task farming engine*; the *scheduler* that consists of a Grid explorer for resource discovery, a schedule advisor backed with scheduling algorithms and a resource trading manager; a *dispatcher and actuators* for deploying agents on grid resources; and *agents* for managing the execution of Nimrod-G jobs on grid resources.

5.1. The task farming engine

The Nimrod-G task farming engine (TFE) is a persistent and programmable job control agent that maintains the record of experiment (jobs, parameters, tasks, deadline and budget), manages and coordinates with other components. It consists of a database to provide persistence that is accessed through a thin management interface. The TFE is responsible for the parametrization of the experiment,

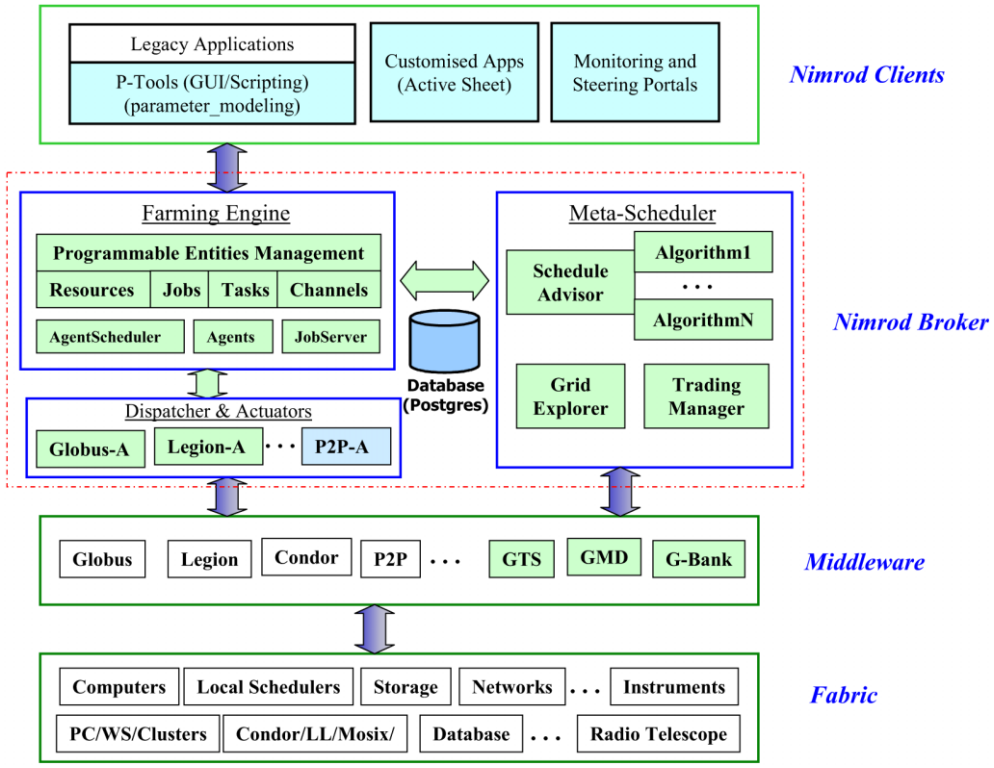


Figure 8. The layered architecture of the Nimrod-G system.

the actual creation of jobs and the maintenance of job status, and it also provides the means for interaction between the clients, the schedule advisor and the dispatcher. The TFE interacts with the scheduler and dispatcher in order to process jobs. It manages the experiment under the direction of schedule advisor, and then instructs the dispatcher to deploy jobs on selected resources and manages their execution.

The TFE maintains the state of an entire experiment and ensures that it is recorded in persistent storage. This allows the experiment to be restarted if the root node fails. The TFE exposes interfaces for job, resource and task management along with the job-to-resource mapping APIs. Accordingly, scheduling policy developers can use these interfaces to implement other schedulers without concern for the complexity of low-level remote execution mechanisms.

The programmable capability of the TFE enables 'plugging' of user-defined schedulers and customized clients or problem-solving environments (e.g., ActiveSheets [36]) in place of the default components. The TFE is a coordination point for processes performing resource trading, scheduling, data and executable staging, remote execution and result collation.

5.2. The scheduler

The scheduler is responsible for resource discovery, resource trading, resource selection and job assignment. The resource discovery algorithm interacts with an information service (the MDS in Globus), identifies the list of authorized and available machines, trades for resource *access cost* and keeps track of resource status information. The resource selection algorithm is responsible for selecting those resources that meet the *deadline and budget* constraints along with optimization requirements. Nimrod-G provides three different scheduling algorithms [7].

5.3. The dispatcher and actuators

The dispatcher triggers appropriate actuators to deploy agents on Grid resources and to assign one of the resource-mapped jobs for execution. Even though the schedule advisor creates a schedule for the entire duration based on user requirements, the dispatcher deploys jobs on resources periodically depending on load and the number of CPUs that are available. We have implemented different dispatchers and actuators for each different middleware service. For example, a Globus-specific dispatcher is required for Globus resources and a Legion-specific component for Legion resources.

5.4. Agents

Nimrod-G agents are deployed on Grid resources dynamically at runtime depending on the scheduler's instructions. The agent is responsible for setting up the execution environment on a given resource for a job. It is responsible for transporting the code and data to the machine, and starting the execution of the task on the assigned resource and sending results back to the TFE. Since the agent operates on the 'far side' of the middleware resource management components, it provides error detection for the user's task, sending the information back to the TFE.

The Nimrod-G agent also records the amount of resource consumed during job execution, such as the CPU time and wall clock time. The online measurement of the amount of resource consumed by the job during its execution helps the scheduler evaluate resource performance and change the schedule accordingly. Typically, there is only one type of agent for all mechanisms, irrespective of whether they are fork or queue nodes. However, different agents are required for different middleware systems.

6. SCHEDULING EXPERIMENTS

We have performed deadline and budget constrained (DBC) scheduling experiments on the World-Wide Grid (WWG) [37] testbed shown in Figure 9. The testbed has computational resources owned by different organizations distributed across five continents: Asia, Australia, Europe, North America and South America. It contains heterogeneous resources such as PCs, workstations, SMPs, clusters and vector supercomputers running operating systems such as Linux, Sun Solaris, IBM AIX, SGI IRIX and Compaq Tru64. Furthermore, the systems use a variety of job management systems such as OS-Fork, NQS, Condor, RMS, PBS and LSF. Most of these resources support secure remote access through the Globus system and a Linux cluster at Virginia is managed using the Legion system. The Solaris workstation from where this scheduling experiment is performed runs Globus, Legion and Condor

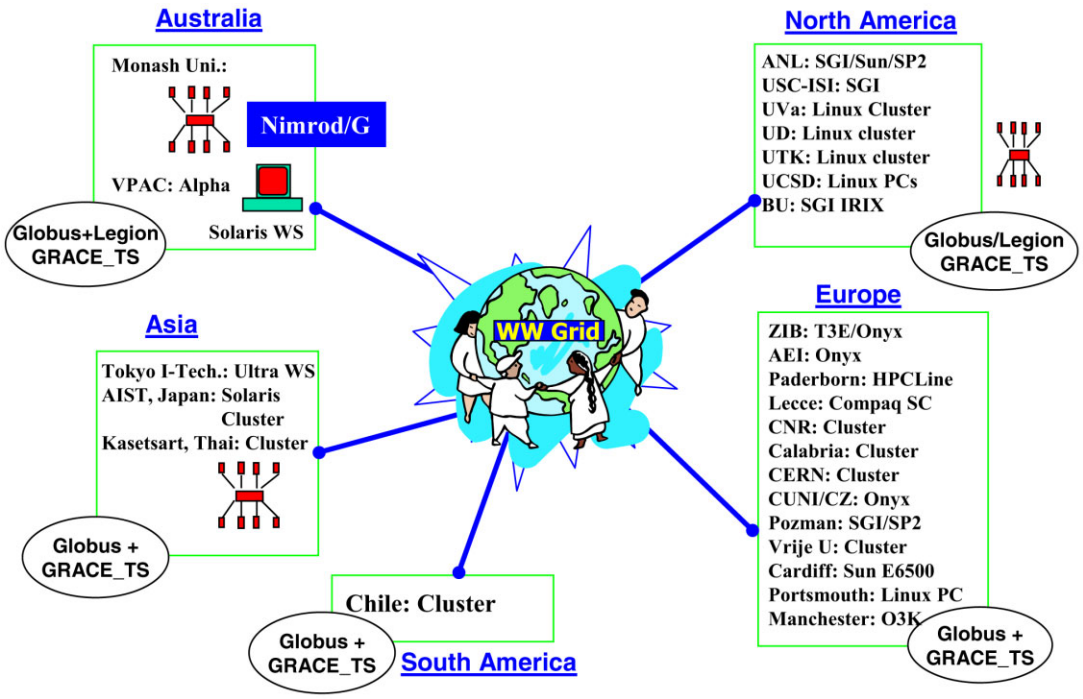


Figure 9. The WWG testbed.

systems along with the Nimrod-G resource broker. The Nimrod-G agents are deployed at runtime on resources for managing the execution of jobs.

We have created a hypothetical parameter sweep application (PSA) that executes a CPU intensive program with 200 different parameter scenarios or values. The program *calc* takes two input parameters and saves results into a file named 'output'. The first input parameter `angle_degree` represents the value of the angle in degrees for processing trigonometric functions. The program *calc* needs to be explored for angular values from 1 to 200 degrees. The second parameter `time_base_value` indicates the expected calculation complexity in minutes plus 0 to 60 seconds positive deviation. This means the program calc is modelled to execute for anywhere between 10 to 11 minutes randomly on resources. A plan for modelling this application as a parameter sweep application using the Nimrod-G parameter specification language is shown in Figure 10. The first part defines parameters and the second part defines the task that needs to be performed for each job. As the parameter `angle_degree` is defined as a range parameter type with values varying from 1 to 200 in steps of 1, it leads to the creation of 200 jobs with 200 different input parameter values. To execute each job on a Grid resource, the Nimrod-G resource broker, depending on its scheduling strategy, first copies the program


```
#Parameters Declaration
parameter angle_degree integer range from 1 to 200 step 1;
parameter time_base_value integer default 10;

#Task Definition
task main
   #Copy necessary executables depending on node type
   copy calc.$OS node:calc
   #Executable program with parameter values on remote node
   node:execute ./calc $angle_degree $time_base_value
   #Copy results file to use home node with jobname as extension
   copy node:output ./output.$jobname
endtask
```

Figure 10. The Nimrod-G parameter sweep processing specification.

executable to a Grid node, then executes the program and finally copies results back to the user home node and stores the output with the job number as file extension.

We have conducted two scheduling experiments for a given deadline of 4 hours (with 18 minutes of extension for the second experiment) and a budget of 250 000 (G$ or tokens) with different optimization strategies [7].

1. Optimize for time. This strategy produces results as early as possible, but before a deadline and within a budget limit.
2. Optimize for cost. This strategy produces results by deadline, but reduces cost within a budget limit.

In these scheduling experiments, the Nimrod-G resource broker employed the *commodity market* model to establish a service access price. It used Grid resource trading services to establish a connection with the Grid trader running on resource providers' machines and obtained service prices accordingly. The broker architecture is generic enough to use any of the protocols discussed above to negotiate access to resources and to choose appropriate ones. The access price varies from one consumer to another and from time to time, as defined by the resource owners. Depending on the deadline and the specified budget, the broker develops a plan for assigning jobs to resources. While doing so it carries out dynamic load profiling to learn the ability of resources to execute jobs. Thus, it adapts itself to the changing resource conditions including failure of resources or jobs on the resource. The heuristics-based scheduling algorithms employed by Nimrod-G broker are presented in our early work [7].

We have used a subset of resources of the WWG testbed in these scheduling experimentations. Table II shows the resource properties such as architecture, location and access price, and the middleware systems loaded on them. Snapshots of the Nimrod-G monitoring and steering clients taken immediately after the completion of the application processing are shown in Figures 11 and 12. The resources used in both experiments are (time/space) shared resources with many other independent users. Hence, they were partially available to us, which changed dynamically depending on other users' requirements and priorities. The access price indicated in the table has been established dynamically

Table II. The WWG testbed resources used in scheduling experiments, job execution and costing.

Organization and location	Vendor, resource type, No. of CPU, OS, hostname	Grid services fabric and role	Price (G$/CPU s)	Number of jobs executed	
				TimeOpt	CostOpt
Monash University, Melbourne, Australia	Sun: Ultra-1, 1 node, bezek.dstc.monash.edu.au	Globus, Nimrod-G, CDB Server, Fork (Master node)	—	—	—
VPAC, Melbourne, Australia	Compaq: Alpha, 4 CPU, OSF1, grendel.vpac.org	Globus, GTS, Fork (Worker node)	1	7	59
AIST, Tokyo, Japan	Sun: Ultra-4, 4 nodes, Solaris, hpc420.hpcc.jp	Globus, GTS, Fork (Worker node)	2	14	2
AIST, Tokyo, Japan	Sun: Ultra-4, 4 nodes, Solaris, hpc420-1.hpcc.jp	Globus, GTS, Fork (Worker node)	1	7	3
AIST, Tokyo, Japan	Sun: Ultra-2, 2 nodes, Solaris, hpc420-2.hpcc.jp	Globus, GTS, Fork (Worker node)	1	8	50
University of Lecce, Italy	Compaq: Alpa cluster, OSF1, sierra0.unile.it	Globus, GTS, RMS (Worker node)	2	0	0
Institute of the Italian National Research Council, Pisa, Italy	Unknown: Dual CPU PC, Linux, barbera.cnuce.cnr.it	Globus, GTS, Fork (Worker node)	1	9	1
Institute of the Italian National Research Council, Pisa, Italy	Unknown: Dual CPU PC, Linux, novello.cnuce.cnr.it	Globus, GTS, Fork (Worker node)	1	0	0
Konrad-Zuse-Zentrum Berlin, Berlin, Germany	SGI: Onyx2K, IRIX, 6, onyx1.zib.de	Globus, GTS, Fork (Worker node)	2	38	5
Konrad-Zuse-Zentrum Berlin, Berlin, Germany	SGI: Onyx2K, IRIX, 16 onyx3.zib.de	Globus, GTS, Fork (Worker node)	3	32	7
Charles University, Prague, Czech Republic	SGI: Onyx2K, IRIX, mat.ruk.cuni.cz	Globus, GTS, Fork (Worker node)	2	20	11
University of Portsmouth, U.K.	Unknown: Dual CPU PC, Linux, marge.csm.port.ac.uk	Globus, GTS, Fork (Worker node)	1	1	25
University of Manchester, U.K.	SGI: Onyx3K, 512 node, IRIX, green.cfs.ac.uk	Globus, GTS, NQS, (Worker node)	2	15	12
Argonne National Laboratory, Chicago, U.S.A.	SGI: IRIX lemon.mcs.anl.gov	Globus, GTS, Fork (Worker node)	2	0	0
Argonne National Laboratory, Chicago, U.S.A.	Sun: Ultra-8, Solaris, 8, pitcairn.mcs.anl.gov	Globus, GTS, Fork (Worker node)	1	49	25
		Total experiment cost (G$)		199 968	141 869
		Time to finish experiment (min)		150	258

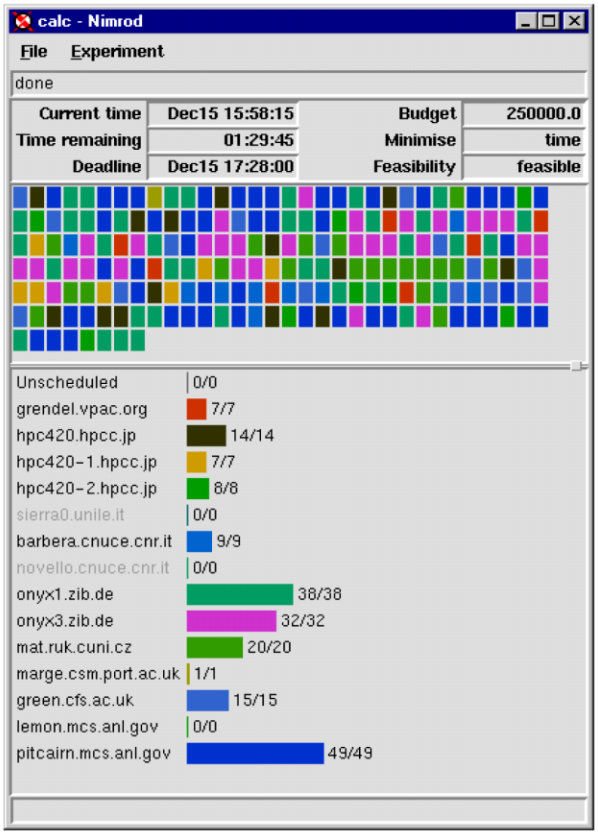


Figure 11. A snapshot of the Nimrod-G monitor during the 'optimize for time' scheduling experiment.

using GRACE resource trading protocols (commodity market model), but is based on an arbitrary assignment by us for demonstration purposes only.

6.1. DBC constrained time optimization scheduling

We performed the first experiment, *optimize for time* scheduling, on 15 December 2001 at 13.28:00, Australian Eastern Daylight Saving Time (AEDT), with 4 hours deadline, and finished on the same day at 15.58:15. A snapshot of the Nimrod-G monitoring and steering client, taken immediately after the completion of the experiment, is shown in Figure 11. This experiment took $2\frac{1}{2}$ hours to finish the processing of all jobs using resources available at that time with an expense of 199 968 G$. Figure 13 shows the number of jobs in execution on different resources and Figure 14 shows the total number of jobs in execution on the Grid during the experiment execution period. It can be observed that during

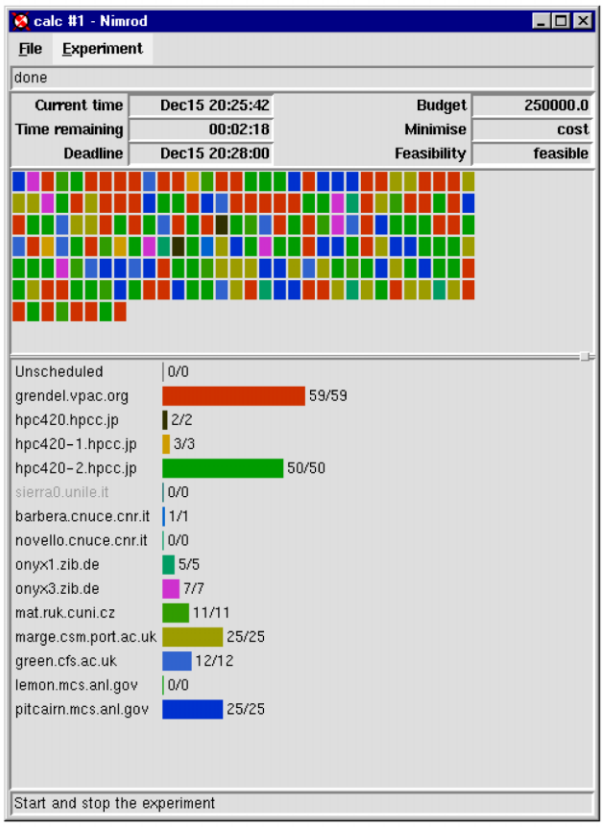


Figure 12. A snapshot of the Nimrod-G monitor during the 'optimize for cost' scheduling experiment.

the first hour of deadline, called the *calibration phase*, the broker aggressively consumed resources for processing jobs to bring the experiment to a feasible state.

Throughout the experiment, two resources, marked in *grey* in Figure 11, were unavailable (either they were shutdown or their resource information servers had failed). We were unable to schedule any jobs on the first ANL node, lemon.mcs.anl.gov, as that was overloaded with jobs from other users. Figure 15 shows the number of jobs processed on different resources selected depending on their cost and availability. Figure 16 shows the total number of jobs processed on the Grid. Figure 17 shows the corresponding expenses of processing on different resources and Figure 18 shows the aggregated processing expenses. From the graphs it can be observed that the broker selected resources to ensure that the experiment was completed at the earliest possible time given the current availability of resources and the budget limitations. It continued to use expensive resources even after the calibration phase deepening on the amount of remaining budget. Another interesting pattern to be observed in

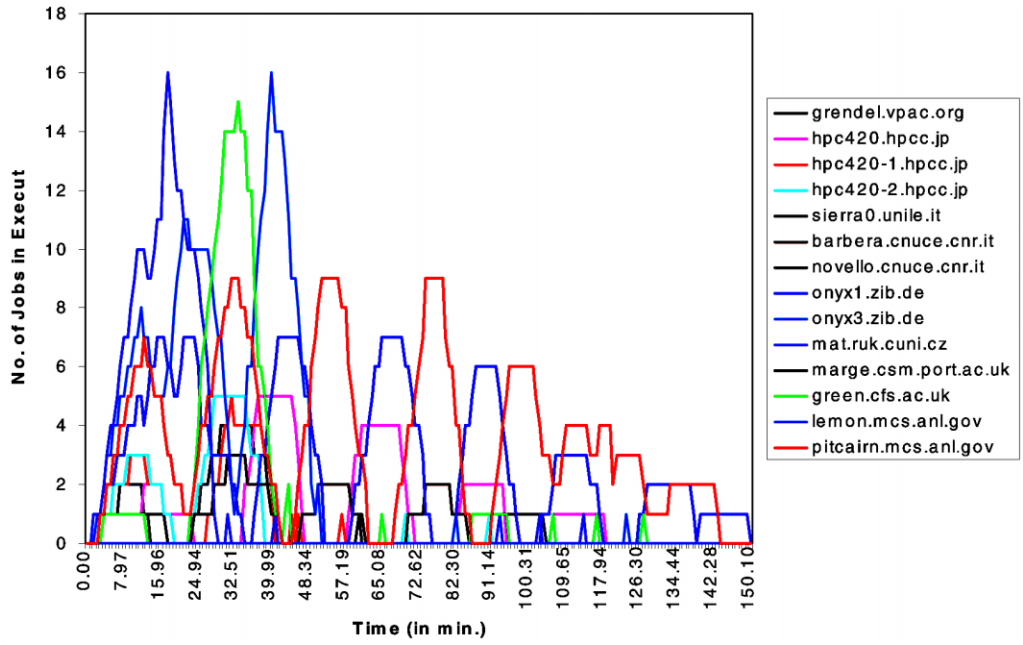


Figure 13. Number of jobs in execution on different Grid resources during DBC time optimization scheduling.

Figures 15 and 17 is that the amount of budget consumed by a resource is not always in proportion to the consumption amount and jobs processed. The most expensive machine (3 G$/s), onyx3.zib.de, was used to process less jobs compared to the other two machines, onyx1.zib.de and pitcairn.mcs.anl.gov, but we had to spend more budget to perform processing on it.

6.2. DBC constrained cost optimization scheduling

The second experiment, *optimize for cost* scheduling, was performed on 15 December 2001 at 16.10:00, AEDT, with 4 hours deadline, which was extended by 20 minutes after 3 hours of execution time. It finished on the same day at 20.25:52. A snapshot of the Nimrod-G monitoring and steering client, taken immediately after the completion of the experiment, is shown in Figure 12. This experiment took 4 hours and 18 minutes to finish the processing of all jobs using resources available at that time with an expense of 141 869 G$. Even though it took more time compared to the first experiment, it saved 58 099 G$.

Figure 19 shows the number of jobs in execution on different resources and Figure 20 shows the total number of jobs in execution on the Grid during the experiment execution period. It can be observed that during the first half-hour, called the *calibration phase*, the broker aggressively consumed resources for processing jobs to bring the experiment to a feasible state. During the experiment, one Italian resource,

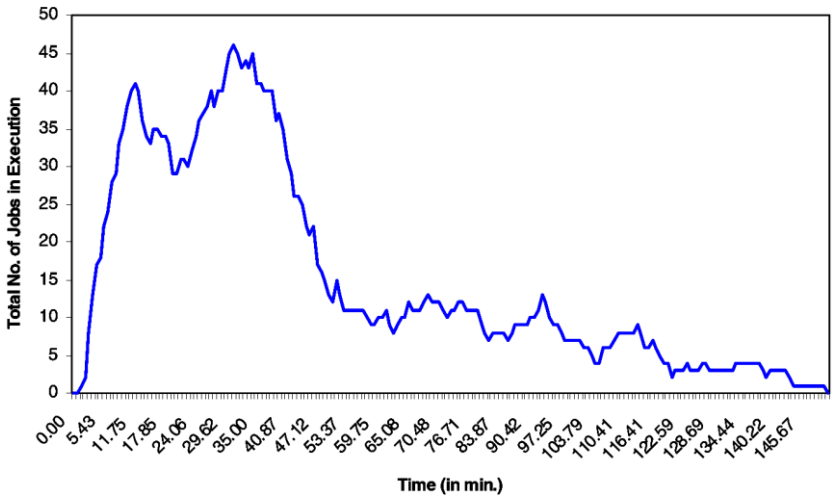


Figure 14. Total number of jobs in execution on the Grid during DBC time optimization scheduling.

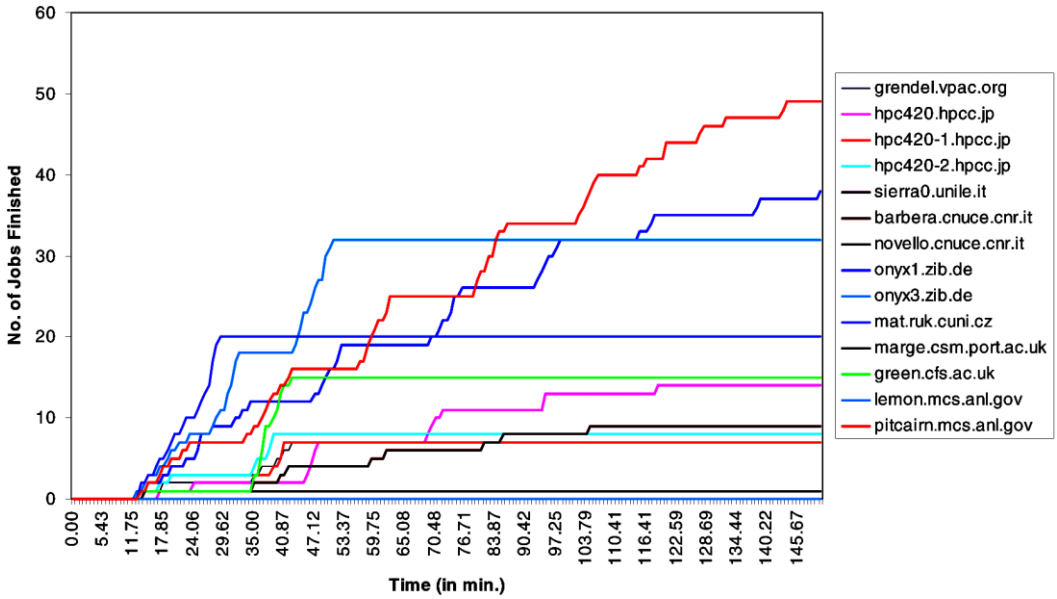


Figure 15. Number of jobs processed on different Grid resources during DBC time optimization scheduling.

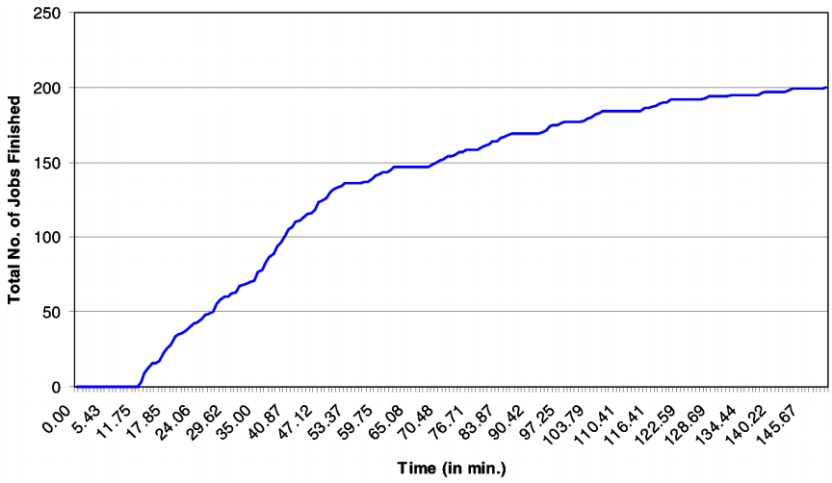


Figure 16. Total number of jobs processed on the Grid resources during DBC time optimization scheduling.

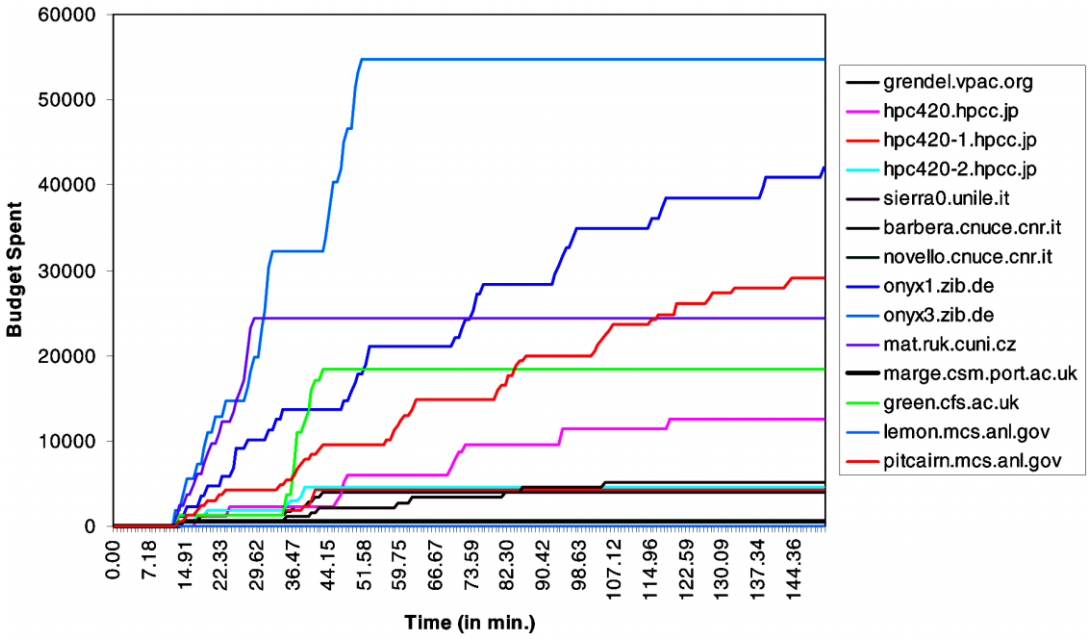


Figure 17. The amount spent on different Grid resources during DBC time optimization scheduling.

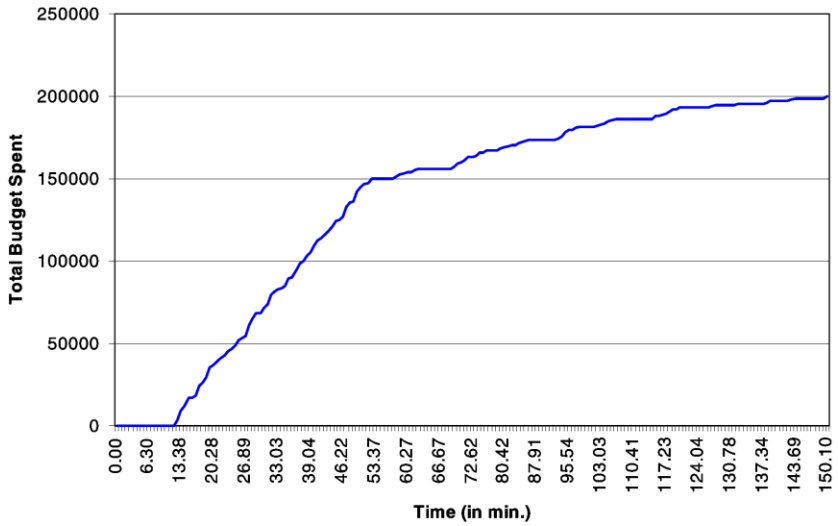


Figure 18. The total amount spent on the Grid during DBC time optimization scheduling.

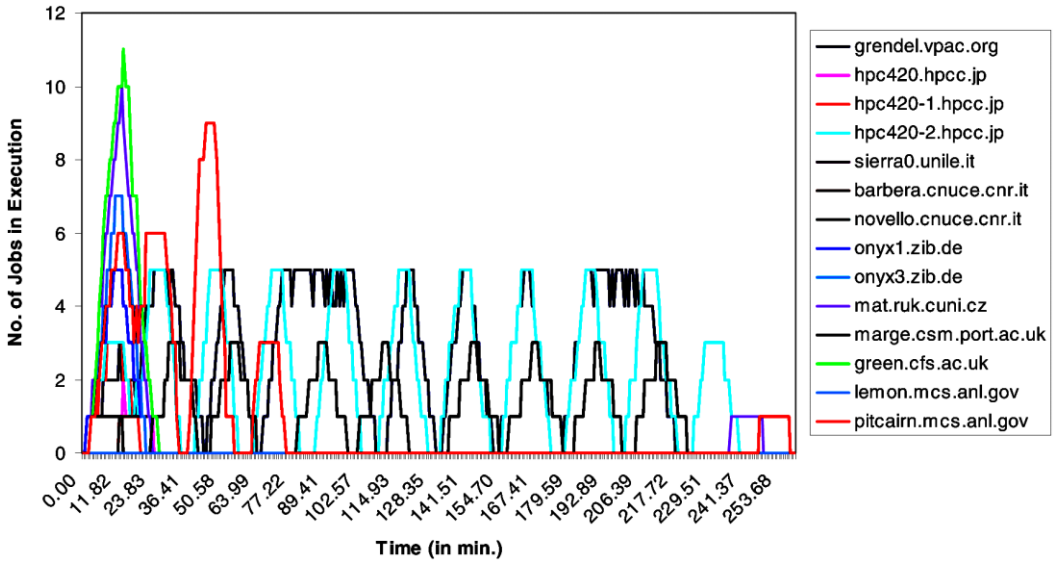


Figure 19. Number of jobs in execution on different Grid resources during DBC cost optimization scheduling.

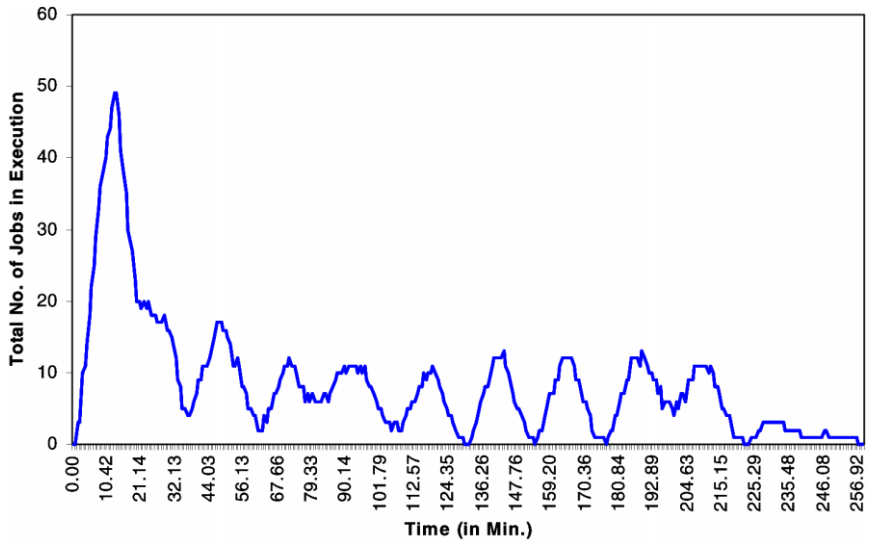


Figure 20. Total number of jobs in execution on the Grid during DBC cost optimization scheduling.

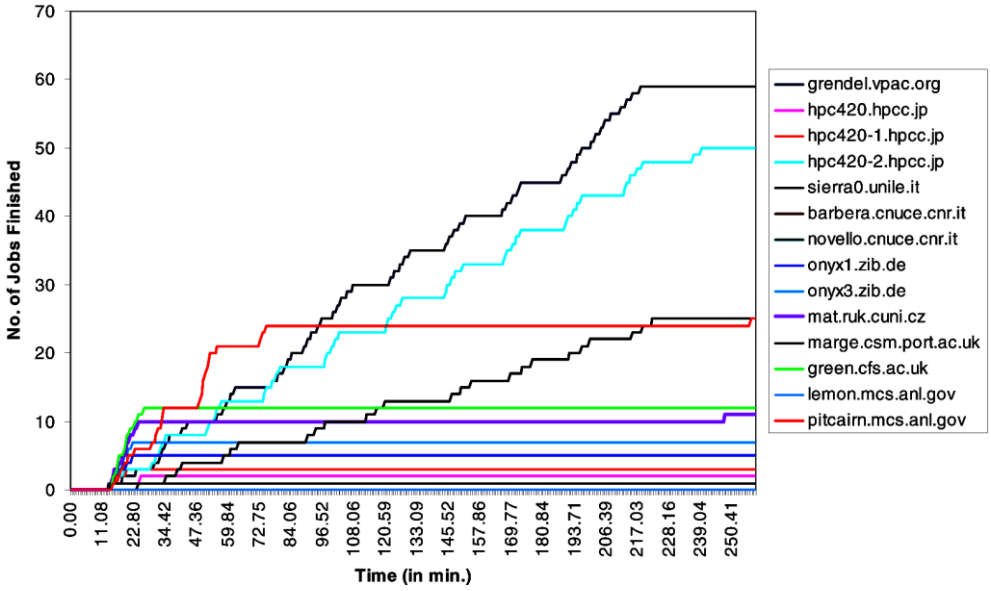


Figure 21. Number of jobs processed on different Grid resources during DBC cost optimization scheduling.

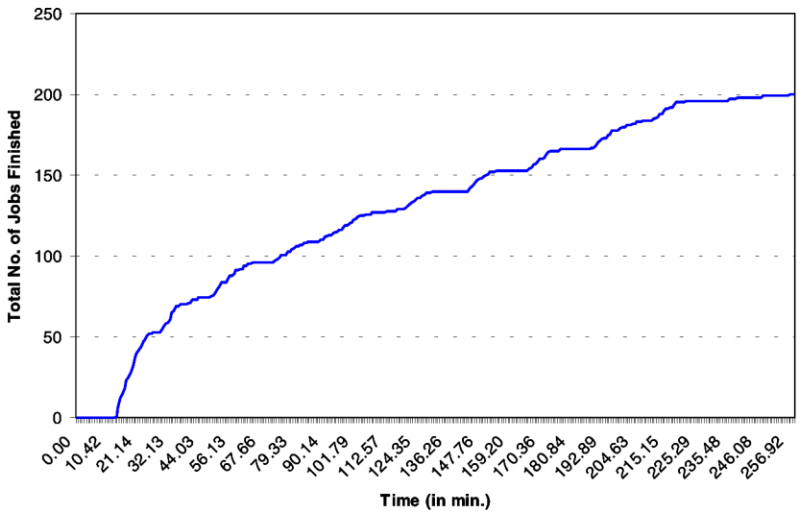


Figure 22. Total number of jobs processed on the Grid resources during DBC cost optimization scheduling.

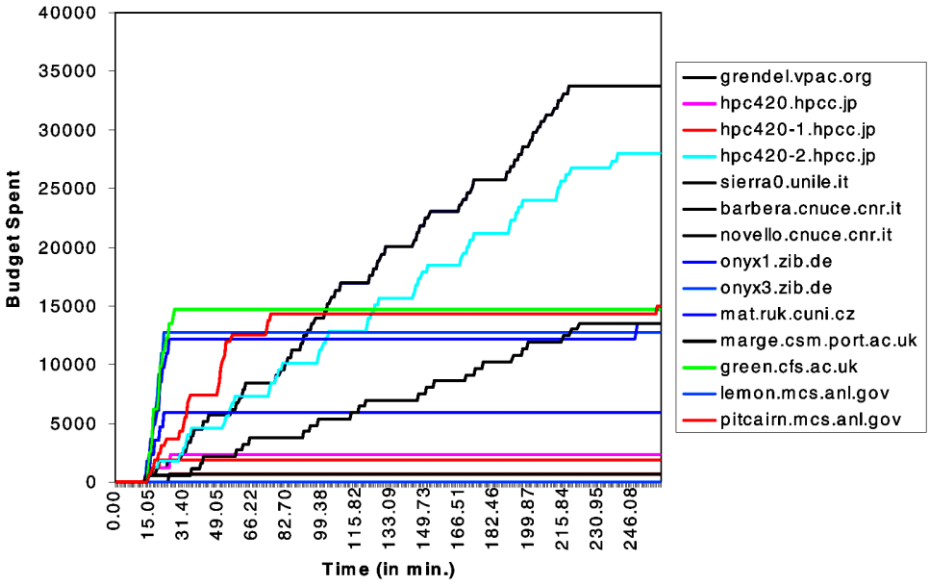


Figure 23. The amount spent on different Grid resources during DBC time optimization scheduling.

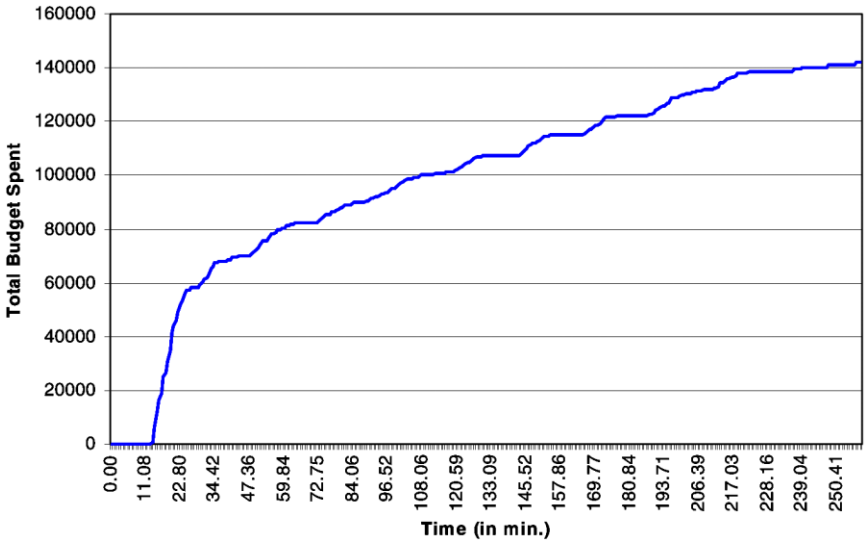


Figure 24. The total amount spent on the Grid during DBC time optimization scheduling.

marked in *grey* in Figure 12, was unavailable. As in the first experiment, we were unable to schedule any jobs on the first ANL node, lemon.mcs.anl.gov, as that was overloaded with jobs from other users. Figure 21 shows the number of jobs processed on different resources selected depending on their cost and availability, and Figure 22 shows the aggregation of jobs processing on the Grid at different times during the experimentation. Figures 23 and 24 show the corresponding amount of budget spent on processing on individual resources and the Grid, respectively.

From the graphs it can be observed that, after the calibration phase, the broker selected the cheapest and most powerful resources extensively to process jobs, as cost minimization was the top priority as long as the deadline could be met. However, it did use a moderately expensive resource, for example, resource mat.ruk.cuni.cz costing 2 G$/CPU s, to process one job to ensure that deadline was met and this was essential as the availability of the cheapest resources had changed from the forecasted availability. As in the first experiment, it can be observed that the amount of budget consumed by a resource was not always in proportion to the number of jobs processed (see Figures 21 and 23) since we had to spend a higher amount for processing on expensive resources.

7. SUMMARY AND CONCLUSION

Computational Grids enable the creation of a virtual computing environment for sharing and aggregation of distributed resources for solving large-scale problems in science, engineering and commerce. The resources in the Grid are geographically distributed and owned by multiple

organizations with different usage and cost policies. They have a large number of self-interested entities (distributed owners and users) with different objectives, priorities and goals that vary from time to time. The management of resources in such a large and distributed environment is a complex task. We proposed the use of computational economy as a metaphor for the management and regulation of supply-and-demand for resources. It provides a decentralized resource management capability and is adaptable to changes in the environment and user requirements. It is a scalable, controllable, measurable and easily understandable policy for management of resources. We discussed several market-based economic models such as a commodity market, tenders and auctions along with the architecture and algorithms for their implementation in Grid computing systems.

We briefly presented a computational economy-driven Grid system called Nimrod-G. The Nimrod-G resource broker supports the deadline and budget constrained algorithms for scheduling task farming applications on large-scale distributed systems. The results of scheduling applications with different QoS requirements on the WWG resource demonstrates the power of these economic models in optimal and effective usage of resources. More importantly, it provides mechanisms to trade-off QoS parameters, deadline and computational cost, and offers an incentive for relaxing their requirements—reduced computational cost for relaxed deadline when the time frame for earliest results delivery is not too critical. We believe this approach of providing an economic incentive for resource owners to share their resources and resource users to trade-off between their deadline and budget promotes the Grid as a platform for mainstream computing, which can lead to the emergence of a new service-oriented computing industry.

ACKNOWLEDGEMENTS

We would like to thank Rob Gray (DSTC) for proof reading and commenting on improving the paper. We acknowledge colleagues, collaborators and organizations (mentioned in the first column of Table II) for providing access to their resources used in the scheduling experimentations reported in the paper.

REFERENCES

1. Abramson D, Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: Killer application for the Global Grid? *Proceedings International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico, 1–5 May 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.
2. Buyya R, Abramson D, Giddy J. A case for economy Grid architecture for service-oriented Grid computing. *Proceedings of the International Parallel and Distributed Processing Symposium: 10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, 23 April 2001, San Francisco, CA. IEEE Computer Society Press: Los Alamitos, CA, 2001.
3. Sullivan III WT, Werthimer D, Bowyer S, Cobb J, Gedye D, Anderson D. A new major SETI project based on Project Serendip data and 100,000 personal computers. *Proceedings of the 5th International Conference on Bioastronomy*, 1997. http://setiathome.ssl.berkeley.edu/woody_paper.html.
4. Ferguson D, Nikolaou C, Sairamesh J, Yemini Y. Economic models for allocating resources in computer systems. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific Press: Singapore, 1996.
5. Buyya R, Abramson D, Giddy J. Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid. *Proceedings 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000)*, Beijing, China, 14–17 May 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.
6. Buyya R, Abramson D, Giddy J. An economy driven resource management architecture for global computational power Grids. *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, 26–29 June 2000, Las Vegas. CSREA Press, 2000.

7. Buyya R, Giddy J, Abramson D. An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications. *Proceedings of the 2nd International Workshop on Active Middleware Services (AMS 2000)*, Pittsburgh, PA, 1 August 2000. Kluwer Academic Press, 2000.
8. Chapin S, Karpovich J, Grimshaw A. The Legion resource management system. *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, San Juan, Puerto Rico, 16 April 1999. Springer: Berlin, 1999.
9. Litzkow M, Livny M, Mutka M. Condor—a hunter of idle workstations. *Proceedings 8th International Conference of Distributed Computing Systems (ICDCS 1988)*, San Jose, CA, January 1988. IEEE Computer Society Press: Los Alamitos, CA, 1988.
10. Berman F, Wolski R. The AppLeS Project: A status report. *Proceedings of the 8th NEC Research Symposium*, Berlin, Germany, May 1997.
11. Casanova H, Dongarra J. NetSolve: A network server for solving computational science problems. *International Journal of Supercomputing Applications and High Performance Computing* 1997; **11**(3):212–223.
12. Kapadia N, Fortes J. PUNCH: An architecture for Web-enabled wide-area network-computing. *Cluster Computing: The Journal of Networks, Software Tools and Applications* 1999; **2**(2):153–164.
13. Brooke J, Foster M, Pickles S, Taylor K, Hewitt T. Mini-Grids: Effective test-beds for Grid application. *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (GRID 2000)*, Bangalore, India, 17 December 2000. Springer: Berlin, 2000.
14. Czajkowski K, Foster I, Karonis N, Kesselman C, Martin S, Smith W, Tuecke S. A resource management architecture for metacomputing systems. *Proceedings of the 4th International Workshop on Job Scheduling Strategies for Parallel Processing*, 30 March 1998, Orlando, FL. Springer: Berlin, 1998.
15. Stonebraker M, Devine R, Kornacker M, Litwin W, Pfeffer A, Sah A, Staelin C. An economic paradigm for query processing and data migration in Mariposa. *Proceedings 3rd International Conference on Parallel and Distributed Information Systems*, Austin, TX, 28–30 September 1994. IEEE Computer Society Press: Los Alamitos, CA, 1994.
16. Heiser G, Lam F, Russell S. Resource management in the Mungi single-address-space operating system. *Proceedings of Australasian Computer Science Conference*, Perth, Australia, 4–6 February 1998. Springer: Singapore, 1998.
17. Nisan N, London L, Regev O, Camiel N. Globally distributed computation over the Internet—the POPCORN project. *International Conference on Distributed Computing Systems (ICDCS'98)*, Amsterdam, The Netherlands, 26–29 May 1998. IEEE Computer Society Press: Los Alamitos, CA, 1998.
18. Amir Y, Awerbuch B, Borgstrom RS. A cost-benefit framework for online management of a metacomputing system. *Proceedings of the 1st International Conference on Information and Computational Economy*, Charleston, SC, 25–28 October 1998.
19. Amir Y, Awerbuch B, Barak A, Borgstrom S, Keren A. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Transactions on Parallel and Distributed Systems* 2000; **11**(7):760–768.
20. Lalis S, Karipidis A. An open market-based framework for distributed computing over the Internet. *Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (GRID 2000)*, Bangalore, India, 17 December 2000. Springer: Berlin, 2000.
21. Reed D, Pratt I, Menage P, Early S, Stratford N. Xenoservers: Accounted execution of untrusted code. *Proceedings 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Rio Rico, AZ, 28–30 March 1999. IEEE Computer Society Press: Los Alamitos, CA, 1999.
22. Bredin J, Kotz D, Rus D. Utility driven mobile-agent scheduling. *Technical Report CS-TR98-331*, Dartmouth College, Hanover, NH, 3 October 1998.
23. Chun B, Culler D. Market-based proportional resource sharing for clusters. *Technical Report CSD-1092*, University of California, Berkeley, CA, January 2000.
24. Mojo Nation. http://www.mojonation.net/ [June 2001].
25. Waldspurger C, Hogg T, Huberman B, Kephart J, Stornetta W. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering* 1992; **18**(2):103–117.
26. Sandholm T. Distributed rational decision making. *Multi-Agent Systems: A Modern Introduction to Distributed Artificial Intelligence*, Weiss G (ed.). MIT Press, 2000.
27. McKnight LW, Boroumand J. Pricing Internet services: Approaches and challenges. *IEEE Computer* 2000; **33**(2): 128–129.
28. Norskog L. A personal communication on economics and Grid allocation, Enron Broadband Systems, 14 March 2001. http://www.buyya.com/ecogrid/comments/enron.txt.
29. Smith R, Davis R. The contract Net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computers* 1980; **29**(12):1104–1113.
30. Vickrey W. Counter-speculation, auctions, and competitive sealed tenders. *Journal of Finance* 1961; **16**(1):9–37.
31. Reynolds K. The Double Auction, Agorics, Inc., 1996. http://www.agorics.com/Library/Auctions/auction6.html.
32. Das R, Hanson J, Kephart J, Tesauro G. Agent–human interactions in the continuous double auction. *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, 4–10 August 2001, Seattle, WA.

33. Hoschek W, Jaen-Martinez J, Samar A, Stockinger H, Stockinger K. Data management in an international data Grid project. *Proceedings of the First IEEE/ACM International Workshop on Grid Computing (GRID 2000)*, Bangalore, India, 17 December 2000. Springer: Berlin, 2000.
34. Stockinger H, Stockinger K, Schikuta E, Willers I. Towards a cost model for distributed and replicated data stores. *Proceedings 9th Euromicro Workshop on Parallel and Distributed Processing (PDP 2001)*, Italy, 7–9 February 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
35. Buyya R, Branson K, Giddy J, Abramson D. The Virtual Laboratory: Enabling molecular modelling for drug design on the World Wide Grid. *Technical Report Monash-CSSE 2001-103*, Monash University, Melbourne, Australia, December 2001.
36. Abramson D, Roe P, Kotler L, Mather D. ActiveSheets: Super-computing with spreadsheets. *Proceedings of the 2001 High Performance Computing Symposium (HPC'01): Advanced Simulation Technologies Conference*, Seattle, WA, 22–26 April 2001.
37. Buyya R. World Wide Grid testbed. http://www.buyya.com/ecogrid/wwg/ [June 2001].

