17

THE NIMROD/G GRID RESOURCE BROKER FOR ECONOMICS-BASED SCHEDULING

RAJKUMAR BUYYA AND DAVID ABRAMSON

17.1 INTRODUCTION

Computational Grids enable the coordinated and aggregated use of geographically distributed resources, often owned by autonomous organizations, for solving large-scale problems in science, engineering, and commerce. However, application composition, resource management, and scheduling in these environments are complex undertakings [18,30]. This is due to the geographic distribution of resources that are often owned by different organizations having different usage policies and cost models, and varying loads and availability patterns. To address these resource management challenges, we have developed a distributed computational economy framework for quality-of-service (Qos)-driven resource allocation and regulation of supply and demand for resources. The new framework offers incentive to resource owners to participate in the Grid and motivates resource users to trade off between time for results delivery and economic cost, namely, deadline and budget [19].

Resource management systems need to provide mechanisms and tools that realize the goals of both service providers and consumers. Resource consumers need a utility model, representing their resource demand and preferences, and brokers that automatically generate strategies for choosing providers on the basis of this model. Further, the brokers need to manage as many issues associated with the execution of the underlying application as possible.

Market-Oriented Grid and Utility Computing Edited by Rajkumar Buyya and Kris Bubendorfer

TABLE 17.1 Economics Models and Their Use in Some Distributed Computing Scheduling Systems

Economic Model	Adopted by
Commodity market	Mungi [9], MOSIX [29], Nimrod/G [20]
Posted price	Nimrod/G
Bargaining	Mariposa [15], Nimrod/G
Tendering or contract-net model	Mariposa
Auction model	Spawn [2], Popcorn [17]
Bid-based proportional resource sharing	Rexec and Anemone [1]
Community and coalition	Condor and SETI@Home [28]
Cooperative bartering	MojoNation [16]
Monopoly and oligopoly	Nimrod/G broker can be used to choose between resources offered at different quality and prices

A computational economy offers many advantages in this environment, because it allows producers and consumers to dynamically negotiate a level of service quality that suits them both. Moreover, when there are multiple users with conflicting demands, they can negotiate access to resources (and thus response time) by "trading" units of currency. A computational economy gives clients a common currency in an otherwise totally distributed system. Service providers benefit from price generation schemes that increase system utilization, as well as economic protocols that help them offer competitive services. For the market to be competitive and efficient, coordination mechanisms that help the market reach an equilibrium price are required; that is, the market price at which the supply of a service equals the quantity demanded [8]. Numerous economic theories have been proposed in the literature, and many commonly used economic models for selling goods and services can be employed as negotiation protocols in Grid computing. Some of these market or social driven economic models are shown in Table 17.1 along with the identity of the distributed system that adopted the approach [21].

These economic models regulate the supply and demand for resources in Grid-based virtual enterprises. We demonstrate the power of these models in scheduling computations using the Nimrod/G resource broker on a large global Grid testbed, called the World Wide Grid (WWG). While it is not the goal of the system to earn revenue for the resource providers, this approach does provide an economic incentive for resource owners to share their resources on the Grid. Further, it encourages the emergence of a new service-oriented computing industry. Importantly, it provides mechanisms to trade off QoS parameters, deadline, and computational cost, and offers incentive for users to relax their requirements. For example, a user may be prepared to accept a later deadline if the computation can be performed at a lower cost.

The rest of this chapter explores the use of an economic paradigm for Grid computing with particular emphasis on providing the tools and mechanisms that support economics-based scheduling. The emphasis will be placed on the Nimrod/G resource broker that supports soft-deadline and budget-based scheduling of parameter

sweep applications [18,32]. Depending on the users' quality-of-service (QoS) requirements, the resource broker dynamically leases Grid services at runtime depending on their cost, quality, and availability. The broker supports the optimization of time or cost within specified deadline and budget constraints. The results of a series of scheduling experiments that we conducted on the WWG testbed using the Nimrod broker will be reported.

17.2 THE NIMROD/G GRID RESOURCE BROKER

17.2.1 Objectives and Goals

Nimrod/G [20,31] is a tool for automated modeling and execution of parameter sweep applications (parameter studies) over global computational Grids [3–7]. It provides a simple declarative parametric modeling language for expressing parametric experiments. A domain expert can easily create a plan for a parametric experiment and use the Nimrod/G system to deploy jobs on distributed resources for execution. It has been used for a very wide range of applications over the years, ranging from quantum chemistry [32] to policy and environmental impact [33]. Moreover, it uses novel resource management and scheduling algorithms based on economic principles. Specifically, it supports user-defined deadline and budget constraints for schedule optimisations and manages supply and demand of resources in the Grid using a set of resource-trading services [19].

Nimrod/G provides a persistent and programmable task-farming engine (TFE) that enables "plugging" of user-defined schedulers and customized applications or problem-solving environments (e.g., ActiveSheets) in place of default components. The task-farming engine is a coordination point for processes performing resource trading, scheduling, data and executable staging, remote execution, and result collation. The Nimrod/G project builds on the early work [5,7] that focused on creating tools that help domain experts compose their legacy serial applications for parameter studies and run them on computational clusters and manually managed Grids. The Nimrod/G system automates the allocation of resources and application scheduling on the Grid using economic principles in order to provide some measurable quality of service (QoS) to the end user. Thus, the focus of this work is within an intersection area of Grid architectures, economic principles, and scheduling optimizations (see Fig. 17.1), which is essential for pushing the Grid into the mainstream computing.

17.2.2 Services and End Users

The Nimrod/G system provides tools for creating parameter sweep applications and services for management of resources and scheduling applications on the Grid. It supports a simple declarative programming language and associated portal and GUI tools for creating scripts and parameterization of application input data files, and a Grid resource broker with programmable entities for scheduling and deploying

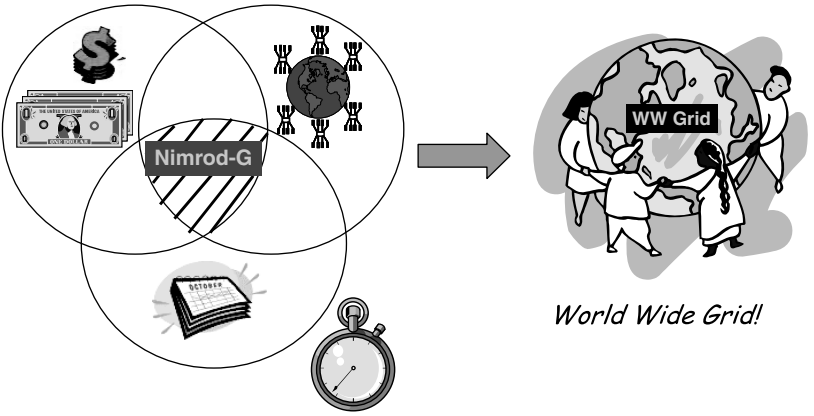

Figure 17.1 QoS-based resource management: intersection of economic, scheduling, and Grid worlds.

jobs on distributed resources. The Nimrod/G resource broker is made up of a number of components—namely, a persistent and programmable task farming engine, a schedule advisor, and a dispatcher—whose functionalities are discussed later. It also provides job management services that can be used for creating user-defined schedulers, steering and monitoring tools, and customized applications. Therefore, the end users that benefit from Nimrod/G tools, protocols, and services are

- *Domain Experts.* This group includes scientific, engineering, and commercial users with large-scale dataset processing requirements. Parameter applications can use Nimrod/G tools to compose them as coarse-grained data-parallel, parameter sweep applications for executing on distributed resources. They can also take advantage of the Nimrod/G broker features to trade off between a deadline and the cost of computation while scheduling application execution on the Grid. This quality of service aspect is important to end users, because the results are useful only if they are returned in a timely manner.
- *Problem-Solving Environments Developers.* Application developers can Grid-enable their applications with their own mechanisms to submit jobs to the Nimrod/G resource broker at runtime depending on user requirements for processing on the Grid. This gives them the ability to create applications capable of directly using Nimrod/G tools and job management services, which, in turn, enables their applications for Grid execution.
- *Task Farming or Master-Worker Programming Environments Designers.* These users can focus on designing and developing easy-to-use and powerful application creation primitives for task farming and master-work style programming model, developing translators and application execution environments by taking advantage of Nimrod/G runtime machinery for executing jobs on distributed Grid resources. Other tools, like Nimrod/O [3,4] use the services of Nimrod/G, for example, to launch jobs on Grid resources.

- *Scheduling Researchers.* The scheduling policy developers generally use simulation techniques and tools such as GridSim [14] for evaluating performance of their algorithms. In simulation, it is very difficult to capture the complete property and behavior of a real-world system; hence, evaluation results may be inaccurate. Accordingly, to prove the usefulness of scheduling algorithms on actual systems, researchers need to develop runtime machinery, which is a resource-intensive and time-consuming task. This can be overcome by using Nimrod/G broker programmable capability. Researchers can use Nimrod/G job management protocols and services to develop their own scheduler and associated scheduling algorithms. The new scheduler can be used to run actual applications on distributed resources and then evaluate the ability of scheduling algorithms in optimally mapping jobs to resources.

17.2.3 Architecture

Nimrod/G leverages services provided by Grid middleware systems such as Globus and Legion. The middleware systems provide a set of low-level protocols for secure and uniform access to remote resources, and services for accessing resources information and storage management. The modular and layered architecture of Nimrod/G is shown in Figure 17.2. The Nimrod/G architecture follows an hourglass

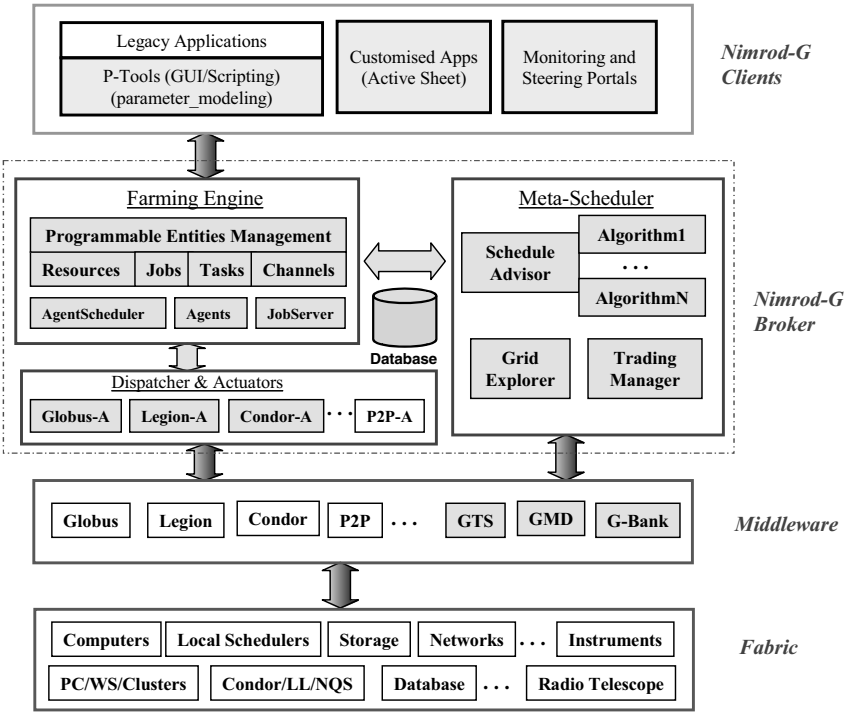

Figure 17.2 A layered architecture of the Nimrod/G system.

design model that allows its implementation on top of different middleware systems and enables the usage of its services by multiple clients and applications.

The key components of Nimrod/G resource broker are:

- Nimrod/G clients, which can be
  - Tools for creating parameter sweep applications
  - Steering and control monitors
  - Customized end-user applications (e.g., ActiveSheets [6])
- The Nimrod/G resource broker, which consists of
  - A task-farming engine (TFE)
  - A scheduler that performs resource discovery, trading, and scheduling
  - A dispatcher and actuator
  - Agents for managing the execution of jobs on resources

The Nimrod/G broker architecture leverages services provided by lower-level different Grid middleware solutions to perform resource discovery, trading, and deployment of jobs on Grid resources.

17.2.4 Nimrod/G Clients

17.2.4.1 Tools for Creating Parameter Sweep Applications. Nimrod supports GUI tools and declarative programming language that assist in creation of parameter sweep applications [7]. They allow the user to (1) parameterize input files; (2) prepare a plan file containing the commands that define parameters and their values; (3) generate a run file, which converts the generic plan file to a detailed list of jobs; and (4) control and monitor execution of the jobs. The application execution environment handles online creation of input files and command line arguments through parameter substitution.

17.2.4.2 Steering and Control Monitors. These components act as a user interface for controlling and monitoring a Nimrod/G experiment. The user can vary constraints related to time and cost that influence the direction the scheduler takes while selecting resources. It serves as a monitoring console and lists the status of all jobs, which a user can view and control. A Nimrod/G monitoring and steering client snapshot is shown in Figure 17.3. Another feature of the Nimrod/G client is that it is possible to run multiple instances of the same client at different locations. This means the experiment can be started on one machine and monitored on another machine by the same or a different user, and the experiment can be controlled from yet another location. We have used this feature to monitor and control an experiment from Monash University and Pittsburgh Supercomputing Centre at Carnegie-Mellon University simultaneously during HPDC-2000 research demonstrations.

17.2.4.3 Customized End-User Applications. Specialized applications can be developed to create jobs at runtime and add jobs to the Nimrod/G engine for

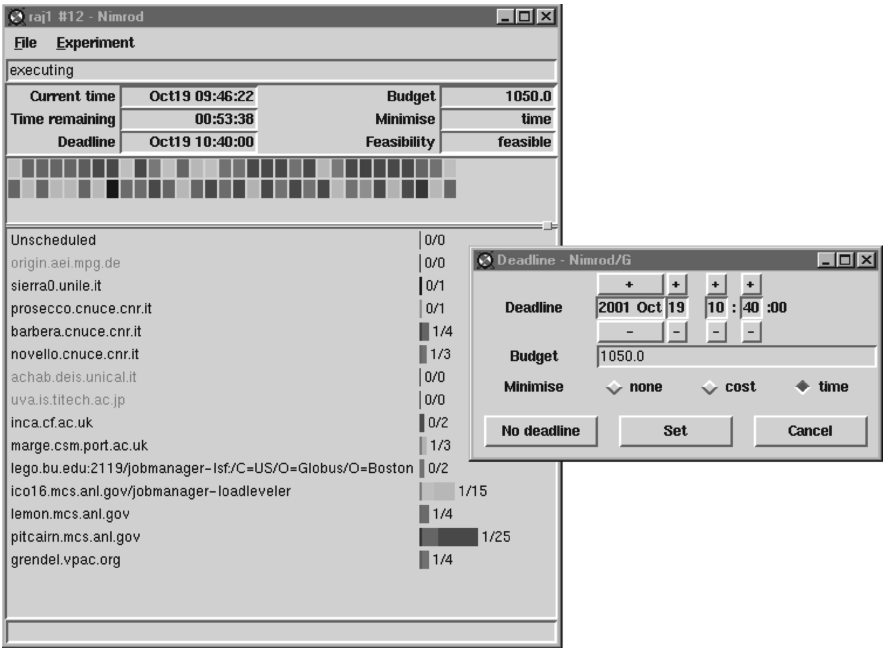

Figure 17.3 A snapshot of Nimrod/G execution monitoring–steering client.

processing on the Grid. These applications can use the Nimrod/G job management services (APIs and protocols described in Ref. 27) for adding and managing jobs. One such application is ActiveSheets [6], an extended Microsoft Excel spreadsheet that submits cell functions as jobs to the Nimrod/G broker for parallel execution on the Grid (see Fig. 17.4). Another example is the Nimrod/O system, a tool that uses nonlinear optimization algorithms to facilitate automatic optimal design [3,4]. This tool has been used on a variety of case studies, including antenna design, smog modeling, durability optimization, airfoil design, and computational fluid dynamics [4].

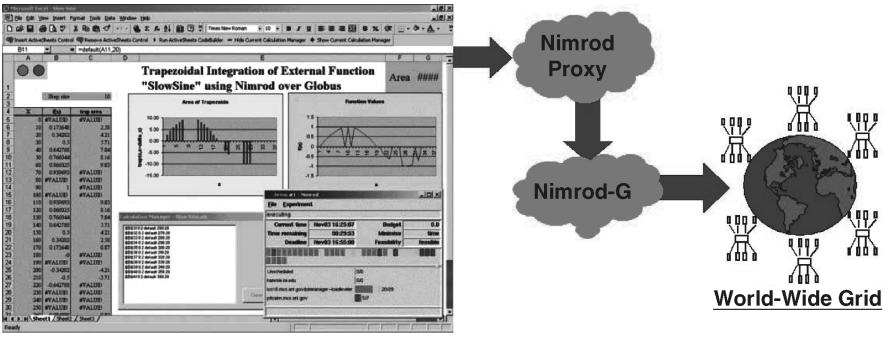

Figure 17.4 ActiveSheet: spreadsheet processing on the Grid using the Nimrod/G broker.

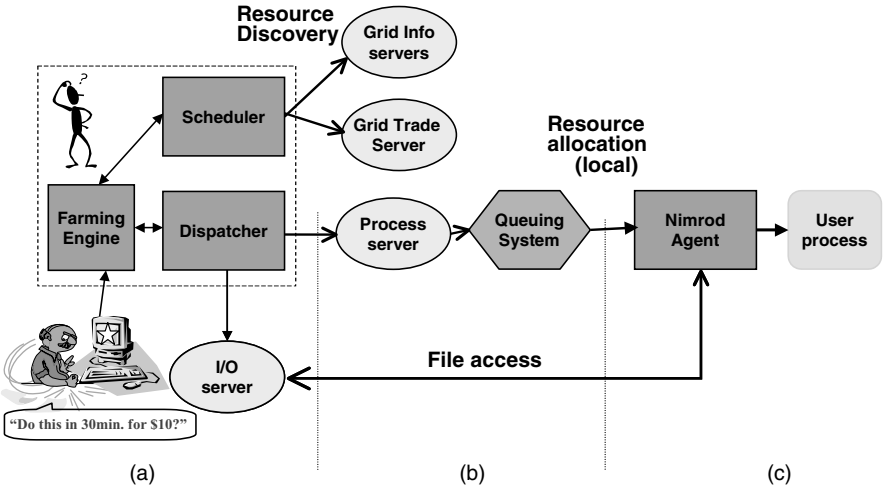

Figure 17.5 Work flow in the Nimrod/G runtime environment: (a) root node; (b) gatekeeper node; (c) computational node.

17.2.5 The Nimrod/G Grid Resource Broker

The Nimrod/G resource broker is responsible for determining the specific requirements that an experiment places on the Grid and performing resource discovery, scheduling, dispatching jobs to remote Grid nodes, starting and managing job execution, and gathering results back to the home node. The submodules of our 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 an actuator for deploying agents on Grid resources, and agents for managing execution of Nimrod/G jobs on Grid resources. The interaction between components of the Nimrod/G runtime machinery and Grid services during runtime is shown in Figure 17.5. The machine on which the broker runs is called the *root node*, the machine (e.g., a cluster master node) that acts as a front end to a Grid resource and forwards the user jobs to a queuing system or forks them for execution is called the *gatekeeper node*, and the machine (e.g., cluster worker node) that executes the user job is called the *computational node*.

17.2.5.1 The Task-Farming Engine (TFE). The Nimrod/G task-farming engine is a persistent and programmable job control agent that manages and controls an experiment. The farming engine is responsible for managing the execution of parameterized application jobs, as well as the actual creation of jobs, the maintenance of job status, and providing a means for interaction between the clients, the schedule advisor, and the dispatcher. The scheduler and dispatcher respectively interact with the TFE to map jobs to resources and deploy on them; that is, the TFE manages the experiment under the direction of schedule advisors,

and then instructs the dispatcher to deploy an application job for execution on the selected resource.

The TFE maintains the state of an entire experiment and ensures that it is recorded in persistent storage. This helps in keeping track of the experiment progress (e.g., status of jobs execution and resources status) and allows the experiment to be restarted if the root node fails without the need for execution of jobs that are already executed. The TFE exposes interfaces for job, resource, and task management along with the job-to-resource mapping APIs and protocols [27]. The developers of scheduling algorithms can use these interfaces to implement their own schedulers rapidly by taking advantage of Nimrod/G TFE and dispatcher capability without concern for the complexity of low-level remote execution mechanisms.

The programmable capability of the task-farming engine enables "plugging" of user-defined schedulers and customized clients or problem-solving environments (e.g., ActiveSheets [6]) in place of the default components. The task-farming engine is a coordination point for processes performing resource trading, scheduling, data and executable staging, remote execution, and result collation.

17.2.5.2 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 metacomputing directory service (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 incorporates three different algorithms (discussed in Section 17.4 for deadline/budget-constrained scheduling [22]).

17.2.5.3 The Dispatcher and Actuators. The dispatcher triggers appropriate actuators—depending on the type of middleware running on resources—to deploy agents on Grid resources and assign one of the resource-mapped jobs for execution. Even though the schedule advisor creates a schedule for the entire duration according to user requirements, the dispatcher deploys jobs on resources periodically, depending on the load and the number of free CPUs available. When the dispatcher decides to deploy computation, it triggers the appropriate actuator depending on middleware service. For example, a Globus-specific actuator is required for Globus resources, and a Legion-specific actuator is required for Legion resources.

17.2.5.4 Agents. Nimrod/G agents are deployed on Grid resources dynamically at runtime depending on the scheduler's instructions. The agent is submitted as a job to the resource process server (e.g., GRAM gatekeeper for a resource running Globus), which then submits to the local resource manager (fork manager in case of time-share resources and queuing system in case of space-shared resource) for starting its execution. The agent is responsible for setting up the execution environment on a given resource for a user job. It is responsible for transporting the code and data to

the machine, 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 needs to provide error detection for the user's job, sending the job termination status 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.

17.3 SCHEDULING AND COMPUTATIONAL ECONOMY

The integration of computational economy as part of a scheduling system greatly influences the way computational resources are selected to meet the user requirements. The users should be able to submit their applications along with their requirements to a scheduling system such as Nimrod/G, which can process the application on the Grid on the user's behalf and try to complete the assigned work within a given deadline and cost. The deadline represents a time by which the user requires the result, and is often imposed by external factors such as production schedules or research deadlines.

To arrive at a scheduling decision, the scheduling system needs to take various parameters into consideration, including the following

- Resource architecture and configuration
- Resource capability (clock speed, memory size)
- Resource state (such as CPU load, memory available, disk storage free)
- Resource requirements of an application
- Access speed (such as disk access speed)
- Free or available nodes
- Priority (that the user has)
- Queue type and length
- Network bandwidth, load, and latency (if jobs need to communicate)
- Reliability of resource and connection
- User preference
- Application deadline
- User capacity/willingness to pay for resource usage
- Resource cost (in terms of dollars that the user need to pay to the resource owner)
- Resource cost variation in terms of timescale (e.g., high at daytime and low at night)
- Historical information, including job consumption rate

The important parameters of computational economy that can influence the way resource scheduling is done are

- Resource cost (set by its owner)
- Price (that the user is willing to pay)
- Deadline (the period by which an application execution needs to be completed)

The scheduler can use the information gathered by a resource discoverer and also negotiate with resource owners to establish service price. The resource that offers the best price and meets resource requirements can eventually be selected. This can be achieved by resource reservation and bidding. If the user deadline is relaxed, the chances of obtaining low-cost access to resources are high. The cost of resources can vary with time, and the resource owner will have the full control over deciding access cost. Further, the cost can vary from one user to another. The scheduler can even solicit bids from resource providers in an open market, and select the feasible service provider(s). To accomplish this, we need scheduling algorithms that take the application processing requirements, Grid resource dynamics, and the user quality-of-service (QoS) requirements such as the deadline, budget, and their optimization preference into consideration. In the next section, we discuss deadline/budget-constrained (DBC) algorithms that we developed for scheduling parameter sweep applications on globally distributed Grid resources.

17.4 SCHEDULING ALGORITHMS

The parameter sweep applications, created using a combination of task and data-parallel models, contain a large number of independent jobs operating different datasets. A range of scenarios and parameters to be explored are applied to the program input values to generate different datasets. The programming and execution model of such applications resemble the single-program multiple-data (SPMD) model. The execution model essentially involves processing N independent jobs (each with the same task specification, but a different dataset) on M distributed computers, where N is, typically, much larger than M.

When the user submits a parameter sweep application containing N tasks along with QoS requirements, the broker performs the following activities:

1. Resource discovery—identifying resources and their properties and then selecting resources capable of executing user jobs.
2. Resource trading—negotiating and establishing service access cost using a suitable economic model.
3. Scheduling—select resources that fit user requirements using *scheduling heuristic/algorithm* and map jobs to them.
4. Deploy jobs on resources (dispatcher).
5. Monitor and steer computations.

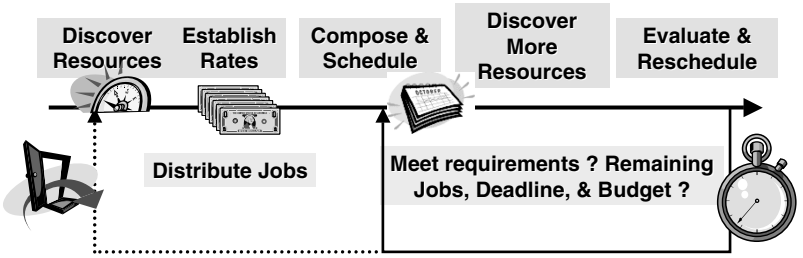

Figure 17.6 High-level steps for adaptive scheduling used in the Nimrod/G broker.

6. Perform load profiling for future usage.
7. When the job execution is finished, gather results back to the user home machine (dispatcher).
8. Record all resource usage details for payment processing purpose.
9. Perform rescheduling—repeat steps 3–8 until all jobs are processed and the experiment is within the deadline and budget limit.
10. Perform cleanup and postprocessing, if required.

The high-level steps for scheduling with deadline and budget constraints are shown in Figure 17.6.

The scheduling and orchestration of the execution of parameter sweep applications on worldwide distributed computers appear simple, but complexity arises when users place QoS constraints such as deadline (execution completion time) and computation cost (budget) limitations. Such a guarantee of service is difficult to provide in a Grid environment since its resources are shared, heterogeneous, distributed in nature, and owned by different organizations having their own policies and charging mechanisms. In addition, scheduling algorithms need to adapt to the changing load and resource availability conditions in the Grid in order to achieve performance and at the same time meet the deadline and budget constraints. In our Nimrod/G application-level resource broker (also called an *application-level scheduler*) for the Grid, we have incorporated three adaptive algorithms for deadline and budget constrained scheduling:

- Cost optimization, within time and budget constraints
- Time optimization, within time and budget constraints
- Conservative time optimization, within time and budget constraints

The role of deadline and budget constraints in scheduling and the objectives of different scheduling algorithms are listed in Table 17.2.

We have developed another new algorithm, called *cost–time optimization scheduling*, which extends the first two (cost–time optimization) scheduling algorithms. This new algorithm and the performance evaluation results are discussed in Section 17.6.

TABLE 17.2 Deadline/Budget-Constrained Scheduling Algorithms and Objectives

Scheduling Algorithm Strategies	Execution Time (Not beyond the Deadline)	Execution Cost (Not beyond the Budget)
Cost optimization	Limited by deadline	Minimize
Time optimization	Minimize	Limited by budget
Conservative time optimization	Limited by deadline	Limited by budget

The *time optimization scheduling* algorithm attempts to complete the experiment as quickly as possible, within the budget available. A description of the core of the algorithm is as follows:

1. For each resource, calculate the next completion time for an assigned job, taking into account previously assigned jobs and job consumption rate.
2. Sort resources by next completion time.
3. Assign one job to the first resource for which the cost per job is less than or equal to the remaining budget per job.
4. Repeat steps 1–3 until all jobs are assigned.

The *cost optimization scheduling* algorithm attempts to complete the experiment as economically as possible within the deadline:

1. Sort resources by increasing cost.
2. For each resource in order, assign as many jobs as possible to the resource, without exceeding the deadline.

The *conservative time optimization* scheduling algorithm attempts to complete the experiment within the deadline and cost constraints, minimising the time when higher budget is available. It spends the budget cautiously and ensures that a minimum of "the budget per job" from the total budget is available for each unprocessed job:

1. Split resources as to whether cost per job is less than or equal to the budget per job.
2. For the cheaper resources, assign jobs in inverse proportion to the job completion time (e.g., a resource with completion time $= 5$ gets twice as many jobs as a resource with completion time $= 10$).
3. For the more expensive resources, repeat all steps (with a recalculated budget per job) until all jobs are assigned.

Note that the implementations of all the algorithms described above contain extra steps for dealing with the initial startup (when the average completion times are unknown), and for when all jobs cannot be assigned to resources (infeasible schedules). Detailed steps of the above mentioned scheduling heuristics are described in Section 17.6.

17.5 IMPLEMENTATION ISSUES AND TECHNOLOGIES USED

The Nimrod/G resource broker follows a modular, extensible, and layered architecture with an "hourglass" principle as applied in the Internet Protocol suite [11]. This architecture enables separation of different Grid middleware systems *mechanisms* for accessing remote resources from the end-user applications. The broker provides uniform access to diverse implementations of low-level Grid services. The key components of Nimrod/G, the task-farming engine, the scheduler, and the dispatcher are loosely coupled. To support the interaction between them, the job management protocols described in Reference 27 have been implemented. Apart from the dispatcher and the Grid Explorer, the Nimrod/G components are independent of low-level middleware used. The modular and extensible architecture of Nimrod/G facilitates a rapid implementation of Nimrod/G support for upcoming peer-to-peer computing infrastructures such as Jxta [12] and Web services [24]. To achieve this, it is necessary to implement only two new components, a dispatcher and an enhanced Grid Explorer. The current implementation of Nimrod/G broker uses low-level Grid services provided by Globus [10] and Legion [25] systems. The Globus toolkit components used in the implementation of Nimrod/G are GRAM (Globus resource allocation manager), MDS (metacomputing directory service), GSI (Globus security infrastructure), and GASS (global access to secondary storage). We also support Nimrod/G dispatcher implementation for Condor [13] resource management system. The use of various Grid and commodity technologies in implementing Nimrod/G components and functionality are listed in Table 17.3.

TABLE 17.3 Nimrod/G Resource Broker Modules Functionality and Role of Grid Services

Nimrod/G Module	Implementation and Grid Technologies Used
Application model	Coarse-grained task farming, master worker, and data parallelism
Application composition	We support mechanism for application parameterization through parameterization of input files and command-line inputs for coarse-grained data parallelism; Nimrod/G basically supports coarse-grain, data-parallel, task farming application model, which can be expressed using our declarative programming language or GUI tools
Application interface	The Nimrod/G broker supports protocols and interfaces [27] for job management; Nimrod/G clients or problem-solving environments can add, remove, and enquire about job status and can set user requirements such as deadline and budget; start/stop application execution at both job and entire-application levels
Scheduling interface	The Nimrod/G broker supports protocols and interfaces [27] for mapping jobs to resources; schedulers can interact with TFE to access user constraints and application jobs details to develop a schedule that maps jobs to resources appropriately
Security	Secure access to resources and computations (identification, authentication, computational delegation) is provided by low-level middleware systems (Globus GSI infrastructure)

TABLE 17.3 *(Continued)*

Nimrod/G Module	Implementation and Grid Technologies Used
Resource discovery	Resource discovery involves discovering appropriate resources and their properties that match the user's requirements; we maintain resource listings for Globus, Legion, and Condor and their static and dynamic properties are discovered using Grid information services; for example, in case of Globus resources, we query Globus LDAP-based GRIS server for resource information
Resource trading and market models	Nimrod/G broker architecture is generic enough to support various economic models for price negotiation and using the same in developing application schedules
Performance prediction	Nimrod/G scheduler performs user-level resource capability measurement and load profiling by measuring and establishing the job consumption rate
Scheduling algorithms	Deadline/budget-based constraint (DBC) scheduling performed by Nimrod/G schedule advisor; Along with DBC scheduling, we support further optimization of time-, cost-, or surplus-driven divide-and-conquer in scheduling
Remote job submission	The Nimrod/G dispatcher performs deployment of Nimrod/G agents using Globus GRAM, Legion, or Condor commands; the agents are responsible for managing all aspects of job execution
Staging programs and data on remote resources	In the case of Legion and Condor, it is handled by their I/O management systems; on Globus resources, we use http protocols for fetching required files
Accounting (broker level)	Nimrod/G agents perform accounting tasks such as measuring resource consumption, and the scheduler performs the entire application-level accounting
Monitoring and steering	Nimrod/G monitoring and steering client
Problem-solving environments	ActiveSheets and Nimrod-O are Grid-enabled using the Nimrod/G broker job management services
Execution testbed	The World Wide Grid (WWG) having resources distributed across five continents

While submitting applications to the broker, user requirements such as deadline and budget constraints need to be set and start application execution. These constraints can be changed at any time during execution. The complete details on application parameterization and jobs management are maintained in the database. In the past the database was implemented as a file-based hierarchical database. In the latest version of Nimrod/G, the TFE database is implemented using a standard "relational" database management system.

The commodity technologies and software tools used in the Nimrod/G implementation include the C and Python programming languages, the Perl scripting language, SQL, and Embedded C for database management. The PostgreSQL database system is used for management of the TFE database and its interaction with other components.

17.6 SCHEDULING EVALUATION ON NIMROD/G SIMULATED TEST QUEUES

In addition to accessing real computational resources, Nimrod can also simulate the execution of jobs on a test queue. These simulated queues are useful for testing the scheduling algorithms, since their behavior can be controlled very precisely. A test queue runs each submitted job in succession, and the apparent wall clock time and reported CPU usage can be controlled exactly. It simulates job execution by waiting for a job length period in "real time," and it is assumed that each test queue has a single CPU. This feature is meant for a simple testing of scheduling algorithms incorporated into the Nimrod/G broker. For a detailed performance evaluation, discrete-event simulation tools such GridSim are used (discussed in the next two sections).

For this simulation, we created experiments containing 100 jobs, each with a 90 s runtime, giving a total computation time of 9000 s. For each experiment, we created 10 test queues with different (but fixed) access costs of 10, 12, 14, 16, 18, 20, 22, 24, 26, and 28 G$/(CPU·s). The optimal deadline for this experiment is achieved when each queue runs 10 jobs in sequence, giving a runtime of 900 s for the 100 jobs.

We selected three deadlines: 990 s (the optimal deadline plus 10%), 1980 s (990×2), and 2970 s (990×3). The 10% allowance allows for the fact that although the queues are simulated, and behave perfectly, the standard scheduler has some delays built in.

We selected three values for the budget. The highest is 252,000 units, which is the amount required to run all jobs on the most expensive queue. Effectively, this allows the scheduler full freedom to schedule over the queues with no consideration for the cost. An amount 171,000 G$ is the budget required to execute 10 jobs on each of the queues. Finally, the lowest budget of 126,000 G$ is the budget required to execute 20 jobs on each of the five cheapest queues. Note that for this value, the deadline of 990 s is infeasible, and the deadline of 1980 s is the optimal deadline plus 10%.

Table 17.4 summarizes results for each combination of scheduling algorithm, deadline and budget, and the resulting percentage of completed jobs, the total runtime, and the final cost. The jobs marked "infeasible" have no scheduling solution that enables 100% completion of jobs. The jobs marked "hard" have only one scheduling solution.

Queue behavior is analyzed by examining queue usage over the period of the experiment. For the cost optimization algorithm, Figure 17.7 shows the node usage for a deadline of 1980s. After an initial spike, during which the scheduler gathers information about the queues, the scheduler calculates that it needs to use the four or five cheapest queues only in order to satisfy the deadline. (Actually, it requires exactly five, but the initial spike reduces the requirements a little.) Note that the schedule is similar, no matter what the allowed budget is. Since we are minimizing cost, the budget plays little part in the scheduling, unless the limit is reached. This appears to have happened for the lowest budget, where the completion rate was 97%. The budget of 126,000 units is only enough to complete the experiment if the five cheapest nodes are used. Because of the initial spike, this experiment appears to have run out of money. The other experiments also did not complete 100% of the jobs, but

TABLE 17.4 Behavior of Scheduling Algorithms for Various Scenarios on Grid

Algorithm	Deadline	Budget ($)	Completed (%)	Time(s)	Cost (G$)	Remarks
Cost optimization	990	126,000	85	946	125,820	Infeasible
	990	171,000	84	942	139,500	Hard
	990	252,000	94	928	156,420	Hard
	1980	126,000	97	1927	124,740	Hard
	1980	171,000	99	1918	128,520	–
	1980	252,000	98	1931	127,620	–
	2970	126,000	98	2931	116,820	–
	2970	171,000	98	2925	116,820	–
	2970	252,000	100	2918	118,800	–
Time optimization	990	126,000	36	955	50,040	Infeasible
	990	171,000	100	913	171,000	Hard
	990	252,000	100	930	171,000	Hard
	1980	126,000	80	1968	101,340	Hard
	1980	171,000	100	909	171,000	–
	1980	252,000	100	949	171,000	–
	2970	126,000	100	2193	126,000	–
	2970	171,000	100	928	171,000	–
	2970	252,000	100	922	171,000	–
Conservative time optimization	990	126,000	78	919	120,060	Infeasible
	990	171,000	99	930	168,480	Hard
	990	252,000	100	941	171,000	Hard
	1980	126,000	97	1902	125,100	Hard
	1980	171,000	100	1376	160,740	–
	1980	252,000	100	908	171,000	–
	2970	126,000	99	2928	125,100	–
	2970	171,000	100	1320	161,460	–
	2970	252,000	100	952	171,000	–

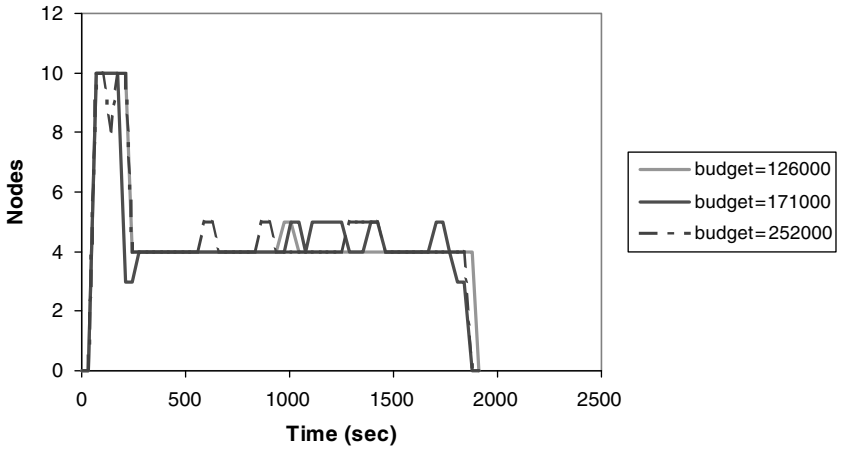

Figure 17.7 DBC cost optimization scheduling algorithm behavior for various budgets.

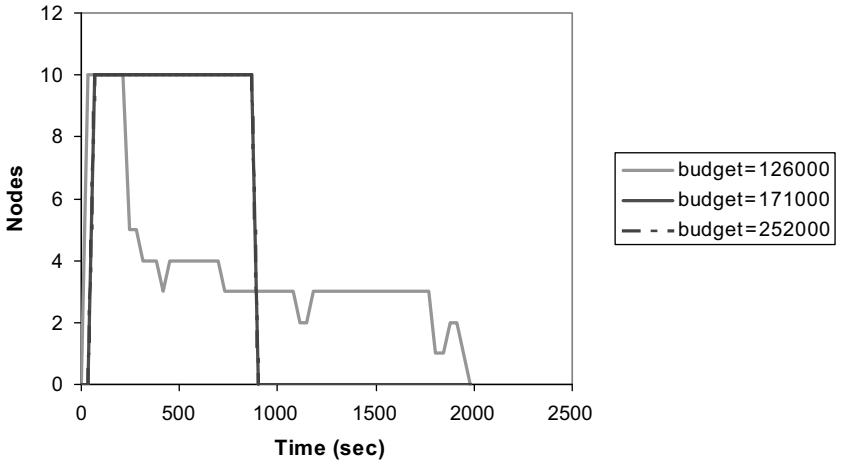

Figure 17.8 Time optimization scheduling algorithm behavior for various budgets.

this is mainly because, in seeking to minimize cost, the algorithm stretches jobs out to the deadline. This indicates the need for a small margin to allow the few remaining jobs to complete close to the deadline.

The equivalent graph for the time optimization algorithm is shown in Figure 17.8. Here we see that, except for the case of a limited budget, we get a rectangular shape, indicating the equal mapping of jobs to each resource. Only the experiment with a very limited budget follows the pattern experienced above.

Looking at the equivalent graph for the conservative time optimization algorithm shown in Figure 17.9, we see much more variation in the schedules chosen for

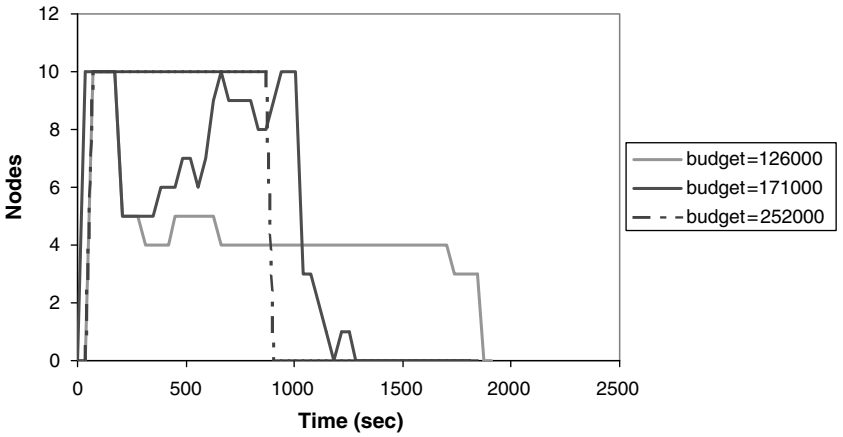

Figure 17.9 Conservative time optimization scheduling algorithm behavior for different budgets.

different budgets. The schedule with a very large budget is equivalent to the time optimization algorithm. The schedule with the low budget is almost the same as the cost optimization algorithm.

17.7 SCHEDULING EXPERIMENTS ON THE WORLDWIDE GRID

We have performed a number of deadline- and budget-constrained scheduling experiments with different requirements at different times by selecting different sets of resources available in the WWG [23] testbed during each experiment. They can be categorized into the following scenarios:

- Cost optimization scheduling during Australian peak and off-peak times
- Cost and time optimization scheduling using cheap local and expensive remote resources

We briefly discuss the WWG testbed followed by a detailed discussion of these scheduling experiments.

17.7.1 The WWG Testbed

To enable our empirical research and experimentations in distributed computational economy and Grid computing, we created and expanded a testbed called the *World Wide Grid* (WWG) in collaboration with colleagues from numerous organizations around the globe. A pictorial view of the WWG testbed depicted in Figure 17.10 shows the name of the organization followed by type of computational resource they have shared. Interestingly, the contributing organizations and the WWG resources themselves are located in five continents: Asia, Australia, Europe, North America, and South America. The organizations whose resources we have used in scheduling experiments reported in this chapter are Monash University (Melbourne, Australia), Victorian Partnership for Advanced Computing (Melbourne, Australia), Argonne National Laboratories (Chicago, USA), University of Southern California's Information Sciences Institute (Los Angeles, USA), Tokyo Institute of Technology (Tokyo, Japan), National Institute of Advanced Industrial Science and Technology (Tsukuba, Japan), University of Lecce (Italy), and CNUCE—Institute of the Italian National Research Council (Pisa, Italy), Zuse Institute Berlin (Berlin, Germany), Charles University, (Prague, Czech Republic), University of Portsmouth (UK), and University of Manchester (UK). In Nimrod/G, these resources are represented using their Internet hostnames.

The WWG testbed contains numerous computers with different architectures, capabilities, and configurations. They include PCs, workstations, SMPs, clusters, and vector supercomputers running operating systems such as Linux, Sun Solaris, IBM AIX (Advanced IBM Unix), SGI IRIX (Silicon Graphics UNIX-like Operating System), and Compaq Tru64. Further, the systems use a variety of job management systems such as OS-Fork, NQS (Network Queueing System),

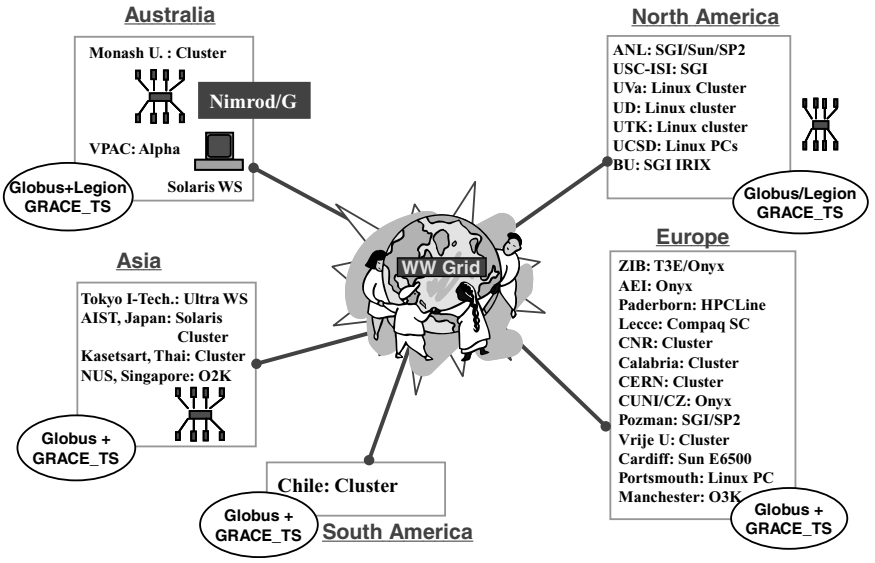

Figure 17.10 The WWG testbed.

Condor, RMS, PBS (Portable Batch System), and LSF (Load Sharing Facility). These system characteristics can be identified by accessing the Grid Information Service (GIS) provided by middleware systems such as Globus running on each resource.

Most of the resources in the WWG testbed support secure remote access through the Globus system and a Linux cluster at Virginia (USA) is managed using the Legion system. The Solaris workstation from where this scheduling experiment is performed runs Globus, Legion, and Condor systems along with the Nimrod/G resource broker. At runtime, the Nimrod/G agents are deployed on resources for managing the execution of jobs.

The properties of WWG testbed resources selected for use in scheduling experiments are discussed in the respective sections. To deploy applications on the Grid using the Nimrod/G broker, the users need to supply the plan that defines application paramterization and task specification, the list of resources that can possibly be utilized, and their QoS requirements such as the deadline, budget, and optimization strategy. The broker discovers the properties of resources using the GIS (e.g., GRIS in the case of Globus) running on them and selects the resources that meet various constraints such as the cost and performance. It also ensures that the application code is available for the target resource architecture. After the selection of resources, the broker maps application jobs to resources using suitable scheduling algorithms. The jobs are then deployed on the Grid by the Nimrod/G dispatcher. To facilitate the tracing of experiments for performance evaluation, the Nimrod/G scheduler records the mapping of jobs and their status at every scheduling event.

Given that the WWG testbed has been used in numerous scheduling experiments with computational economy and real applications (like molecular modeling for drug design), we believe that it truly represents a blueprint of an emerging scalable Grid

computing infrastructure for integrating and aggregating dispersed heterogeneous resources.

17.7.2 Cost Optimization Scheduling—Australian Peak and Off-Peak Times

In a competitive commodity market economy, the resources are priced differently at different times according to the supply and demand. For example, they are priced higher during peak hours and lower during off-peak hours. In this experiment we explore their impact on the processing cost, by scheduling a resource-intensive parameter sweep application containing a large number of jobs on the WWG resources, during Australian peak and off-peak hours.

17.7.2.1 WWG Computational Resources. The WWG testbed resources selected for use in this experiment and their properties are shown in Table 17.5. To test the trading services provided by GTS (Grid trade server), we ran an experiment entirely during peak time and the same experiment entirely during off-peak time. It is important to note access price variations during peak and off-peak times and also time difference between Australia and the United States. The access price is expressed in Grid units (G$) per CPU second.

We selected five resources (see Table 17.5) from the testbed, each effectively having 10 nodes available for our experiment. Monash University has a 60-processor Linux cluster running Condor, which was reduced to 10 available processors for the experiment. Similarly, a 96-node SGI at Argonne National Laboratory (ANL) was made to provide 10 nodes by using *Condor glidein* to add 10 processors to the Condor pool. An 8-node Sun at Argonne and a 10-node SGI at the Information Sciences Institute (ISI) of the University of Southern California were accessed using Globus directly. Argonne's 80-node SP2 was also accessed directly through Globus. We relied on its high workload to limit the number of nodes available to us. We assigned artificial cost (access price per second) for each of those resources depending on their relative capability. This is achieved by setting a resource cost database, which is

TABLE 17.5 Worldwide Grid Testbed Resources Used in the Experiment[a]

Resource Type and Size (Nr. of Nodes)	Organization and Location	Grid Services and Fabric	Price at AU Peak Time	Price at AU Off-Peak Time
Linux cluster (60 nodes)	Monash, Australia	Globus/Condor	20	5
IBM SP2 (80 nodes)	ANL, Chicago, USA	Globus/LL	5	10
Sun (8 nodes)	ANL, Chicago, USA	Globus/Fork	5	10
SGI (96 nodes)	ANL, Chicago, USA	Globus/Condor-G	15	15
SGI (10 nodes)	ISI, Los Angeles, USA	Globus/Fork	10	20

[a] Prices are given in Grid units (G$) and CPU per second.

maintained on each resource by its owner. The resource cost database contains access cost (price) that resource owners like to charge to all their Grid users at different times of the day. The access price generally differs from user to user and time to time.

17.7.2.2 Parameter Sweep Application. We have created a hypothetical parameter sweep application (PSA) that executes a CPU-intensive program with 165 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 angle in degree for processing trigonometric functions. The program `calc` needs to be explored for angular values from 1 to 165°. The second parameter `time_base_value` indicates the expected calculation complexity in minutes plus 0–60 s positive deviation. That means that the program `calc` is expected to run for anywhere between 5 and 6 min on resources with some variation depending on resource capability. A plan file modeling this application as a parameter sweep application using the Nimrod/G parameter specification language is shown in Figure 17.11. 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 165 in step 1, it leads to the creation of 165 jobs with 165 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 executable(s) and necessary data to a Grid node, then executes the program, and finally copies results back to the user home node and stores output with job number as file extension.

17.7.2.3 Scheduling Experiments. The experiments were run twice, once during the Australian peak time, when the US machines were in their off-peak times, and again during the US peak, when the Australian machine was off-peak. The experiments were configured to *minimize the cost*, within a *one-hour deadline*. This requirement instructs the Nimrod/G broker to use the *cost optimization scheduling* algorithm in scheduling jobs for processing on the Grid.

```
#Parameters Declaration
parameter angle_degree integer range from 1 to 165 step 1;
parameter time_base_value integer default 5;

#Task Definition
task main
    #Copy necessary executables depending on node type
    copy calc. $OS node:calc
    #Execute 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 17.11 Nimrod/G parameter sweep processing specification.

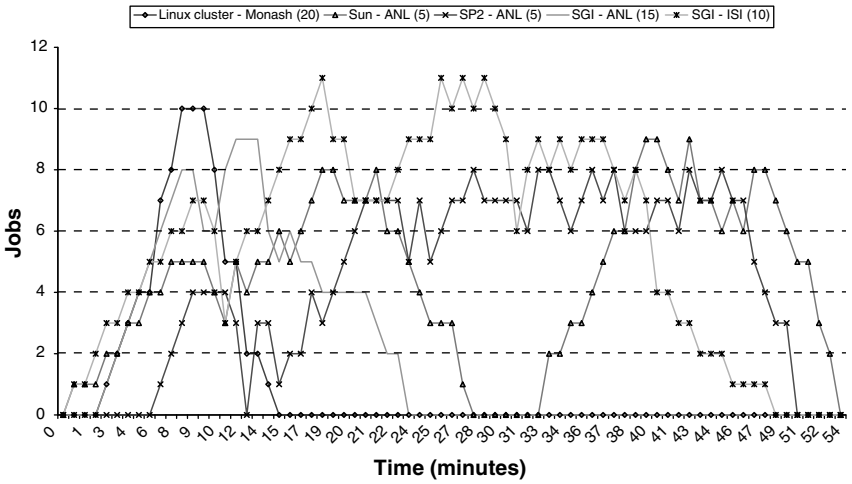

Figure 17.12 Computational scheduling during Australian peak time (US off-peak time).

The number of jobs in execution or queued on resources during the Australian peak and off-peak time scheduling experimentations is shown in Figures 17.12 and 17.13, respectively. The results for the Australian peak experiment show the expected typical results. After an initial calibration phase, the jobs were distributed to the cheapest machines for the remainder of the experiment. This characteristic of the scheduler is clearly visible in both experiments. In the Australian peak experiment, after calibration period, the scheduler excluded the usage of Australian resources as they were expensive and the scheduler predicted that it could still meet the deadline using cheaper resources

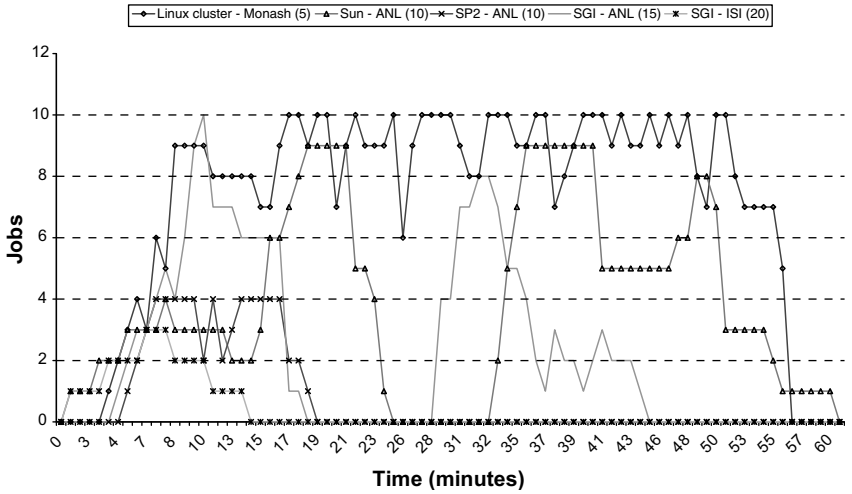

Figure 17.13 Computational scheduling during Australian off-peak time (US peak time).

from US resources, which were in off-peak time phase. However, in the Australian off-peak experiment, the scheduler never excluded the usage of Australian resources and excluded the usage of some of the US resources, as they were expensive comparatively at that time (US in peak-time phase). The results for the US peak experiment are somewhat more interesting (see Fig. 17.13). When the Sun-ANL machine becomes temporarily unavailable, the SP2, at the same cost, was also busy, so a more expensive SGI is used to keep the experiment on track to complete before the deadline.

When the scheduling algorithm tries to minimize the cost, the total cost Australian peak-time experiment is 471,205 G$ and the off-peak time is 427,155 G$. The result is that costs were quite low in both cases. An experiment using all resources, without the cost optimization algorithm during the Australian peak, costs 686,960 G$ for the same workload. The cost difference indicates a saving in computational cost, and it is certainly a successful measure of our budget/deadline-driven scheduling on the Grid.

The number of computational nodes (CPUs) in use at different times during the execution of scheduling experimentation at Australian peak time is shown in Figure 17.14. It can be observed that in the beginning of the experiment (calibration phase), the scheduler had no precise information related to job consumption rate for resources; hence it attempted to use as many resources as possible to ensure that it could meet the deadline. After the calibration phase, the scheduler predicted that it could meet the deadline with fewer resources and stopped using more expensive nodes. However, whenever scheduler senses difficulty in meeting the deadline by using the resources currently in use, it includes additional resources. This process continues until deadline is met and at the same time ensures that the cost of computation is within a given budget.

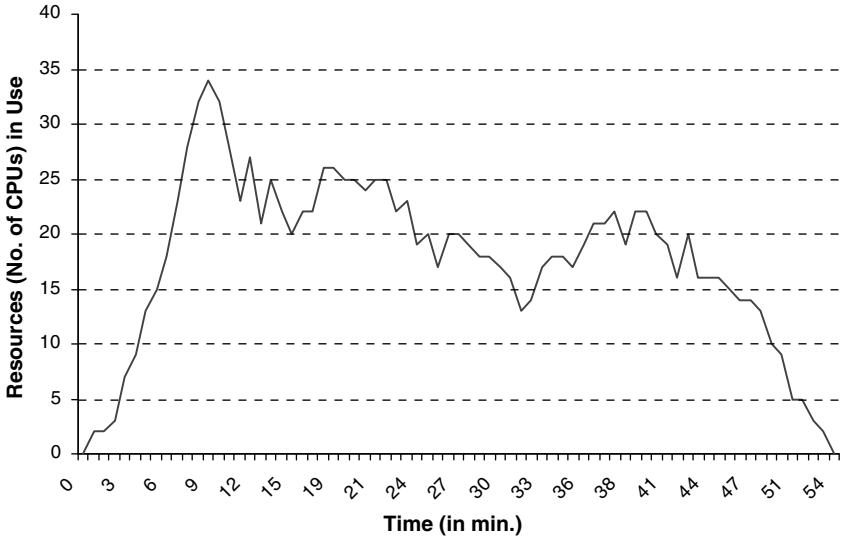

Figure 17.14 Number of resources in use during Australian peak-time scheduling experiment.

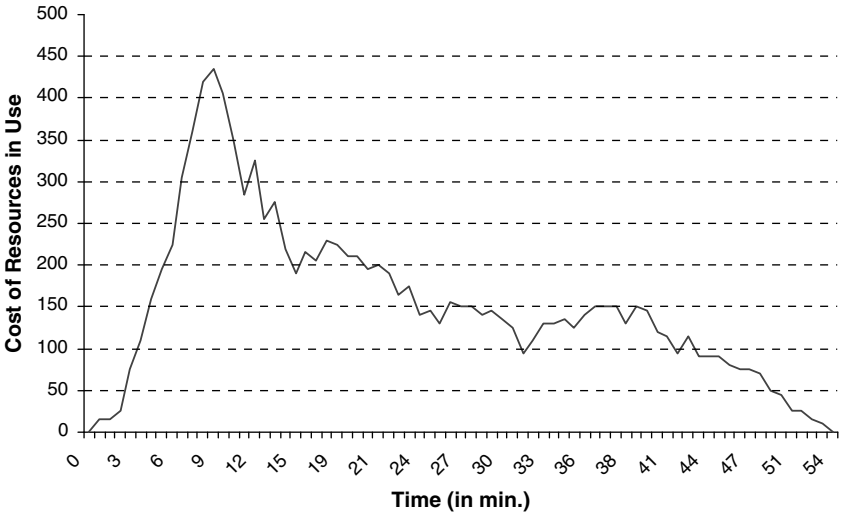

Figure 17.15 Cost of resources in use at Australian peak-time scheduling experiment.

The total cost of resources (sum of the access price for all resources) in use at different times during the execution of scheduling experimentation at Australian peak time is shown in Figure 17.15. It can be observed that the pattern of variation of cost during the calibration phase is similar to that of number of resources in use. However, this is not the same as the experiment progresses, and in fact the cost of resources decreased almost linearly although the number of resources in use did not decline at the same rate. The reason for this behavior is that a large number of resources selected by the scheduler were located in off-peak time zones (i.e., USA was in off-peak time when Australia was in peak hours) as they were less expensive. Another reason is that the number of resources used in these experiments contains more US resources compared to Australian resources.

Similar behavior did not occur in scheduling experiments conducted during Australian off-peak time (see Figs. 17.16 and 17.17). The variation pattern of total number of resources in use and their total cost is similar because the larger numbers of US resources were available cheaply. Although the scheduler has used Australian resources throughout the experiment (see Fig. 17.13), the scheduler had to depend on US resources to ensure that the deadline is met even if resources were expensive.

17.7.3 Cost and Time Optimization Scheduling Using Local and Remote Resources

This experiment demonstrates the use of cheap local resources and expensive remote resources together for processing a parameter sweep application (the same as that used in the previous scheduling experiment) containing 165 CPU-intensive jobs, each running approximately 5 min in duration. We have set the deadline of 2 h (120 mins)

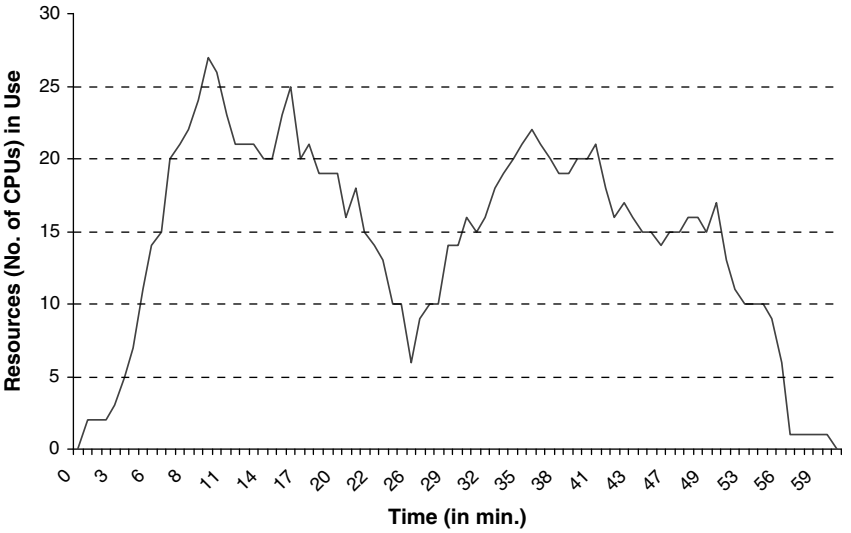

Figure 17.16 Number of resources in use at Australian off-peak time scheduling experiment.

and budget of 396,000 (G$ or tokens) and conducted experiments for two different optimization strategies:

- *Optimize for time*—this strategy produces results as early as possible, but before a deadline and within a budget limit.

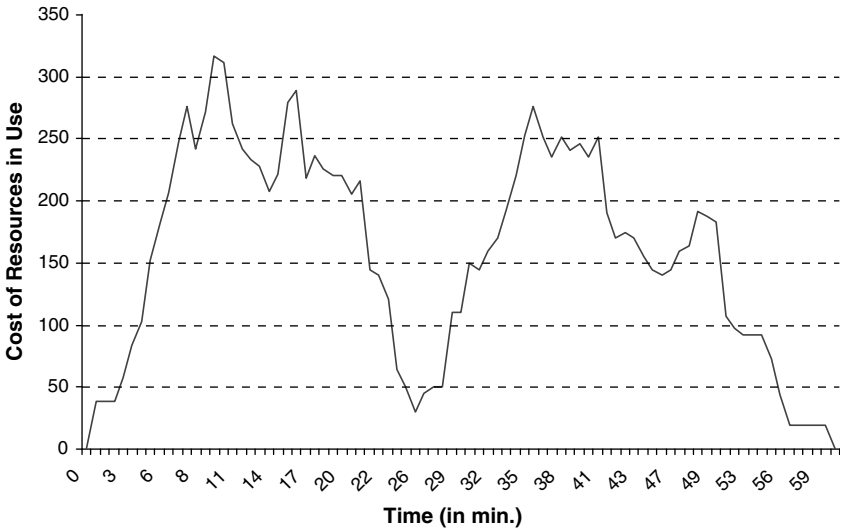

Figure 17.17 Cost of resources in use at Australian off-peak time scheduling experiment.

- *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 for establishing a service access price. The broker established connection with the Grid Trader running on resource providers' machines to obtain service prices at runtime. The broker architecture is generic enough to use any of the protocols discussed by Buyya et al. [21] for negotiating access to resources and choosing appropriate ones. The access price varies for local and remote users; users are encouraged to use local resources since they are available at cheaper price. Depending on the deadline and the specified budget, the broker develops a plan for assigning jobs to resources. While doing so, it does dynamic load profiling to establish the user job consumption rate for each resource. The broker uses this information to adapt itself to the changing resource conditions including failure of resources or jobs on the resource.

We have used a subset of resources of the WWG testbed in these scheduling experiments. Table 17.6 shows resource details such as architecture, location, and

TABLE 17.6 The WWG Testbed Resources Used in Scheduling Experiments, Job Execution, and Costing

Resource Type and Size (Nr. of Nodes)	Organization and Location	Grid Services and Fabric	Price [G$/(CPU:s)]	Jobs Executed on Resources	
				`Time_Opt`	`Cost_Opt`
Linux cluster (60 nodes)	Monash, Australia	Globus, GTS, Condor	2	64	153
Solaris (Ultra-2)	Tokyo Institute of Technology, Japan	Globus, GTS, Fork	3	9	1
Linux PC (Prosecco)	CNUCE, Pisa, Italy	Globus, GTS, Fork	3	7	1
Linux PC (Barbera)	CNUCE, Pisa, Italy	Globus, GTS, Fork	4	6	1
Sun (8 nodes)	ANL, Chicago, USA	Globus, GTS, Fork	7	42	4
SGI (10 nodes)	ISI, Los Angeles, USA	Globus, GTS, Fork	8	37	5
Total experiment cost (G$)				237,000	115,200
Time to complete experiment (min.)				70	119

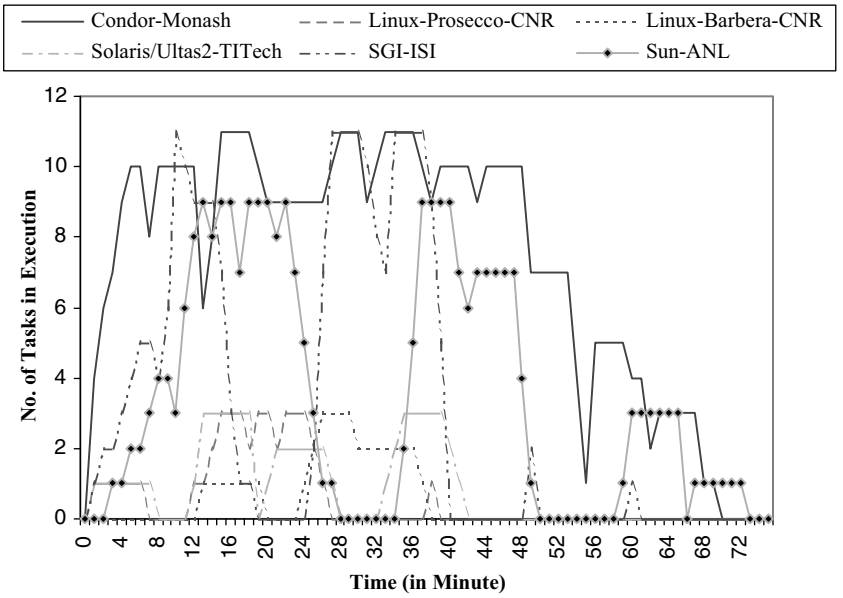

Figure 17.18 Resource selection in deadline and budget constrained time optimization scheduling.

access price along with type of Grid middleware systems used in rendering them Grid-enabled. These are shared resources, and hence they were not fully available to us. The access price indicated in the table is being established dynamically (commodity market model). The access price are artificial; however, they assigned to reflect the offering of differentiated services at different costs as in the real-world marketplace.

The number of jobs in execution on resources (*Y* axis) at different times (*X* axis) during the experimentation is shown in Figures 17.18 and 17.19 for the time and cost optimization scheduling strategies, respectively. In the first (time minimization) experiment, the broker selected resources in such a way that the whole application execution is completed at the earliest time for a given budget. In this experiment, it completed execution of all jobs within 70 min and spent 237,000 G$. In the second experiment (cost minimization), the broker selected cheap resources as much as possible to minimize the execution cost while still trying to meet the deadline (completed in 119 min) and spent 115,200 G$. After the initial *calibration phase*, the jobs were distributed to the cheapest machines for the remainder of the experiment. The processing expense of the time optimization scheduling experiment is much larger than the cost optimization scheduling experiment because of the use of expensive resources to complete the experiment early. The results show that our Grid brokering system can take advantage of economic models and user input parameters to meet their requirements.

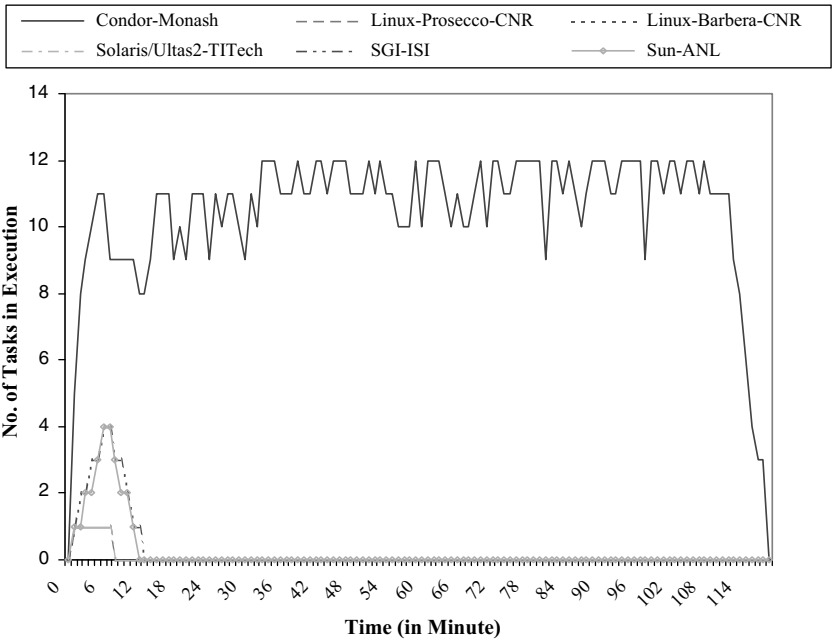

Figure 17.19 Resource selection in deadline/budget-constrained cost optimization scheduling.

17.8 SUMMARY AND COMMENTS

We have discussed the design, development, and experimental evaluation of the Nimrod/G Grid resource broker that supports deadline and budget constrained and quality of service requirements–driven application scheduling on worldwide distributed resources. The broker is able to dynamically adapt itself when there is change in the availability of resources and user QoS requirements during the application execution. It also supports scalable, controllable, measurable, and easily enforceable policies and scheduling algorithms for allocation of resources to user applications. It demonstrates that the computational economy approach for Grid computing provides an effective means for pushing Grids into mainstream computing and enables the creation of a worldwide Grid marketplace.

The Nimrod tools for modeling parametric experiments are mature and in production use for cluster and Grid computing. The Nimrod/G task-farming engine (TFE) services have been used in developing customized clients and applications. An associated dispatcher is capable of deploying computations (jobs) on Grid resources enabled by Globus, Legion, and Condor. The TFE jobs management protocols and services can be used for developing new scheduling policies. We have built a number of market-driven deadline- and budget-constrained scheduling algorithms, namely, time and cost optimizations with deadline and budget constraints. The results of

scheduling experiments with different QoS requirements on the World Wide Grid resource show promising insights into the effectiveness of an economic paradigm for management of resources, and their usefulness in application scheduling with optimizations. The results demonstrate that the users have options and can, indeed, trade off between the deadline and the budget depending on their requirements, thus encouraging them to reveal their true requirements to increase the value delivered by the utility.

ACKNOWLEDGMENTS

The Nimrod project began in 1994 at the Co-operative Research Centre for Distributed Systems Technology (DSTC) in Brisbane, Australia. It has been funded by various Australian government grants, including the Co-operative Research Centres scheme and the Australian Research Council. Today, it continues at Monash University, where it is supported by the Australian Research Council. We would like to thank Jon Giddy for his contribution to development of the Nimrod/G Grid Resource Broker. This chapter is partially derived from earlier publications [18–20,22,30]. We thank Christian Vecchiola for his help with typesetting.

REFERENCES

1. B. Chun, *Market-based Cluster Resource Management*, PhD dissertation, Univ. California Berkeley, Oct. 2001.
2. C. Waldspurger, T. Hogg, B. Huberman, J. Kephart, and W. Stornetta, Spawn: A distributed computational economy, *IEEE Transactions on Software Engineering* **18**(2):103–117 (Feb. 1992).
3. D. Abramson, A. Lewis, and T. Peachy, Nimrod/O: A tool for automatic design optimization, *Proc. 4th International Conf. Algorithms & Architectures for Parallel Processing* (ICA3PP 2000), Hong Kong, China, Dec. 2000.
4. D. Abramson, A. Lewis, Peachy, and C. Fletcher, An automatic design optimization tool and its application to computational fluid dynamics, *Proc. Super Computing 2001 Conf.*, Denver, CO, Nov. 2001.
5. D. Abramson, I. Foster, J. Giddy, A. Lewis, R. Sosic, R. Sutherst, and N. White, The Nimrod computational workbench: A case study in desktop metacomputing, *Proc. Australian Computer Science Conf. (ACSC 97),* Macquarie Univ., Sydney, Feb. 1997.
6. D. Abramson, P. Roe, L. Kotler, and D. Mather, ActiveSheets: Super-computing with spreadsheets, *Proc. 2001 High Performance Computing Symp. (HPC'01), Advanced Simulation Technologies Conf.,* April 2001.
7. D. Abramson, R. Sosic, J. Giddy, and B. Hall, Nimrod: A tool for performing parametised simulations using distributed workstations, *Proc. 4th IEEE International Symp. High Performance Distributed Computing*, Virginia, Aug. 1995, IEEE CS Press, 1995.
8. D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini, Economic models for allocating resources in computer systems, in *Market-Based Control: A Paradigm for Distributed Resource Allocation*, S. H. Clearwater (ed.), World Scientific Press, Singapore, 1996.

9. G. Heiser, F. Lam, and S. Russell, Resource management in the Mungi single-address-space operating system, *Proc. Australasian Computer Science Conf.*, Perth, Australia, Feb. 4–6, 1998, Springer-Verlag, Singapore, 1998.
10. I. Foster and C. Kesselman, Globus: A metacomputing infrastructure toolkit, *International Journal of Supercomputer Applications* **11**(2):115–128 (1997).
11. J. Postel, C. Sunshine, and D. Cohen, The ARPA Internet Protocol, *Computer Networks* **5** (1981).
12. L. Gong, *Project JXTA: A Technology Overview*, Technical Report, Sun Microsystems Inc., April 2001, http://www.jxta.org/project/www/docs/TechOverview.pdf.
13. M. Litzkow, M. Livny, and M. Mutka, Condor—a hunter of idle workstations, *Proc. 8th International Conf. Distributed Computing Systems (ICDCS 1988)*, San Jose, CA, Jan. 1988, IEEE CS Press, 1988.
14. R. Buyya and M. Murshed, GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing, *Concurrency and Computation: Practice and Experience* **14**(13–15):1175–1220 (Nov.–Dec. 2002).
15. M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin, An economic paradigm for query processing and data migration in Mariposa, *Proc. 3rd International Conf. Parallel and Distributed Information Systems*, Austin, TX, Los Alamitos, CA, Sept. 28–30, 1994, IEEE Computer Society Press, 1994.
16. Mojo Nation, http://www.mojonation.net/, June 2001.
17. N. Nisan, S. London, O. Regev, and N. Camiel, Globally distributed computation over the Internet: The POPCORN project, *Proc. International Conf. Distributed Computing Systems (ICDCS'98)*, Amsterdam, The Netherlands, May 26–29, 1998, IEEE CS Press, 1998.
18. R. Buyya, D. Abramson, and J. Giddy, A case for economy Grid architecture for service-oriented Grid computing, *Proc. International Parallel and Distributed Processing Symp.: 10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, San Francisco, CA, April 23, 2001, IEEE CS Press, 2001.
19. R. Buyya, D. Abramson, and J. Giddy, An economy driven resource management architecture for global computational power Grids, *Proc. 2000 International Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, NV, June 26–29, 2000, CSREA Press, 2000.
20. R. Buyya, D. Abramson, and J. Giddy, Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid, *Proc. 4th International Conf. High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, Beijing, China, May 2000, IEEE Computer Society Press.
21. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, Economic models for resource management and scheduling in Grid computing, *Concurrency and Computation: Practice and Experience* **14**(13–15):1507–1542 (2002).
22. R. Buyya, J. Giddy, and D. Abramson, An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications, *Proc. 2nd International Workshop on Active Middleware Services (AMS 2000)*, Kluwer Academic Press, Aug. 1, 2000, Pittsburgh, PA.
23. R. Buyya, The World-Wide Grid (WWG), http://www.buyya.com/ecogrid/wwg/, 1999–2002.
24. S. Bansal and G. Pal,The Web at your (machine's) service, *JavaWorld Magazine* (Sep. 28, 2001), http://www.javaworld.com/javaworld/jw-09-2001/jw-0928-smsservice.html.

25. S. Chapin, J. Karpovich, and A. Grimshaw, The legion resource management system, *Proc. 5th Workshop on Job Scheduling Strategies for Parallel Processing*, San Juan, Puerto Rico, April 16, 1999, Springer-Verlag Press, Germany, 1999.
26. S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, A directory service for configuring high-performance distributed computations, *Proc. 6th IEEE International Symp. High-Performance Distributed Computing (HPDC 1997)*, Portland, OR, Aug. 1997.
27. TurboLinux, Using application programming interface, in *EnFuzion Manual*, 2002, Chapter 9, available at http://www.csse.monash.edu.au/cluster/enFuzion/api.htm.
28. W. T. Sullivan, III, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson, A new major SETI project based on Project Serendip data and 100,000 personal computers, *Proc. 5th International Conf. Bioastronomy*, 1997.
29. Y. Amir, B. Awerbuch., A. Barak A., S. Borgstrom, and A. Keren, An opportunity cost approach for job assignment in a scalable computing cluster, *IEEE Transactions on Parallel and Distributed Systems* **11**(7):760–768 (July 2000).
30. R. Buyya, *Economic-Based Distributed Resource Management and Scheduling for Grid Computing*, PhD thesis, Monash Univ., Melbourne, Australia, 2002.
31. D. Abramson, J. Giddy, and L. Kotler, High performance parametric modeling with Nimrod/G: Killer application for the global Grid?, *Proc. 14th International Parallel and Distributed Processing Symp.* Cancun, Mexico, May 2000, pp. 520–528.
32. W. Sudholt, K. Baldridge, D. Abramson, C. Enticott, and S. Garic, Parameter scan of an effective group difference pseudopotential using Grid computing, *New Generation Computing* **22**(2):125–135 (2004).
33. A. Lynch, D. Abramson, K. Beringer, and P. Uotila, Influence of savanna fire on Australian monsoon season precipitation and circulation as simulated using a distributed computing environment, *Geo*
physical Research Letters **34**(L20801): 1–5 (2007).