Chapter 4

The Nimrod-G Grid Resource Broker and Economic Scheduling Algorithms

This chapter presents the Nimrod-G Grid resource broker as an example of a Grid system that uses a computational economy driven architecture for managing resources and scheduling task farming applications on large-scale distributed resources. It discusses a layered and component-oriented modular architecture for the Nimrod-G broker design and development. The architecture is generic enough to leverage services provided by various Grid middleware systems such as Globus, Legion, and Condor for uniform access to diverse resources; and the GRACE trading mechanisms. It briefly discusses the deadline and budget constrained scheduling algorithms that we developed and incorporated into the Nimrod-G broker. We evaluate the performance of scheduling strategies and their ability in meeting the user quality of service requirements. Finally, we discuss the results of scheduling parameter sweep applications on the World-Wide Grid resources using the Nimrod-G resource broker to demonstrate its ability in allocating resources depending on their availability, capability, user-level performance, and cost.

4.1 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 is a complex undertaking [98]. 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-driven resource allocation and regulation of supply and demand for resources. The new framework offers incentive to resource owners for being part of the Grid and motivates resource users to trade off between time for results delivery and economic cost, i.e., deadline and budget [99].

Resource management systems need to provide mechanisms and tools that realize the goals of both service providers and consumers. The resource consumers need a *utility model*, representing their resource demand and preferences, and *brokers* that automatically generate strategies for choosing providers based on this model. Further, the brokers need to manage all issues associated with the execution of the underlying application. The service providers need *price generation schemes* so as to increase system utilization, as well as economic *protocols* that help them to offer competitive services. For the market to be competitive and efficient, coordination mechanisms are required that help the market reach an equilibrium price, that is, the market price at which the supply of a service equals the quantity demanded [22]. 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 1 along with the identity of the distributed system that adopted the approach [103].

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 Grid testbed, called the World Wide Grid (WWG) spanning across five continents. Whilst 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.

Table 4.1: Economics models and their use in some distributed computing scheduling systems.

Economic Model	Adopted by
Commodity Market	Mungi [33], MOSIX [145],& Nimrod-G [100]
Posted Price	Nimrod-G
Bargaining	Mariposa [83] & Nimrod-G
Tendering or Contract-Net Model	Mariposa
Auction Model	Spawn [14] & Popcorn [87]
Bid-based Proportional Resource Sharing	Rexec & Anemone [9]
Community and Coalition	Condor and SETI@Home [141]
Cooperative Bartering	MojoNation [84]
Monopoly and Oligopoly	Nimrod-G broker can be used to choose between resources offered at different quality and prices.

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 economic-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 on the Grid [98]. 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 optimisation 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.

4.2 The Nimrod-G Resource Broker: An Economic based Grid Scheduler

4.2.1 Objectives and goals

Nimrod-G is a tool for automated modeling and execution of parameter sweep applications (parameter studies) over global computational Grids [100][98]. 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 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 [99].

Nimrod-G provides a persistent and programmable *task-farming engine* (TFE) that enables "plugging" of user-defined schedulers and customised 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 [18][21] that focused on creating *tools* that help domain experts to 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 to the end user. That is, the focus of this thesis is within an intersection area of Grid architectures, economic principles, and scheduling optimizations (see Figure 4.1), which is essential for pushing the Grid into the mainstream computing.

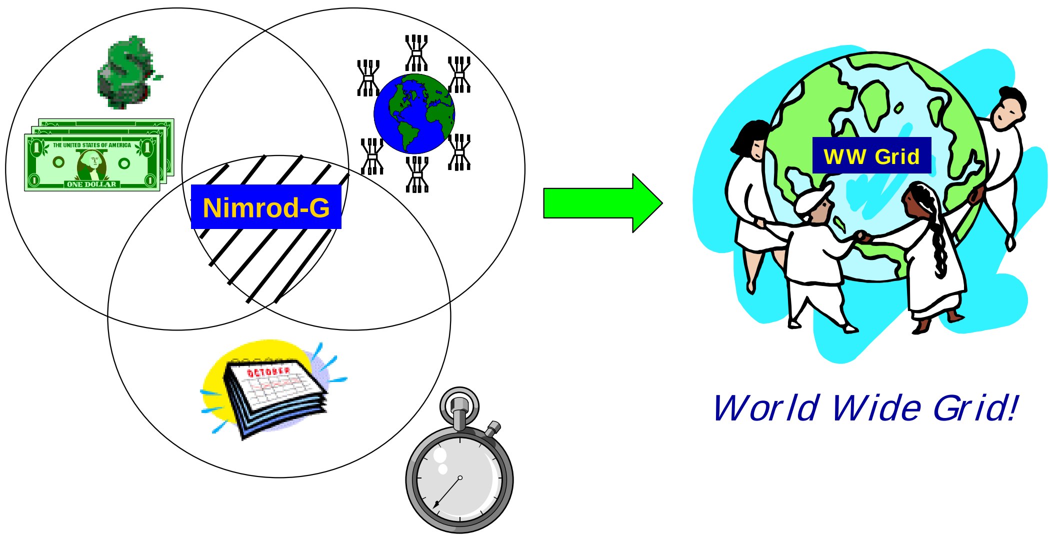


Figure 4.1: QoS based resource management: intersection of economic, scheduling, and Grid worlds.

4.2.2 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 GUI tools for creating scripts and parameterization of application input data files, and a Grid resource broker with programmable entities for scheduling and deploying 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 data-set 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 only useful 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.
- **Scheduling Researchers:** The scheduling policy developers generally use simulation techniques and tools such as GridSim [81] and Simgrid [43] for evaluating performance of their algorithms. In simulation, it is very difficult to capture the complete property and behavior of a real world system and 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.

4.2.3 Architecture

Nimrod-G has been developed as a grid resource broker based on the GRACE framework. It leverages services provided by Grid middleware systems such as Globus, Legion, and the GRACE trading mechanisms. 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 4.2. The Nimrod-G architecture follows hour-glass design model that allows its implementation on top of different middleware systems and enables the usage of its services by multiple clients/applications.

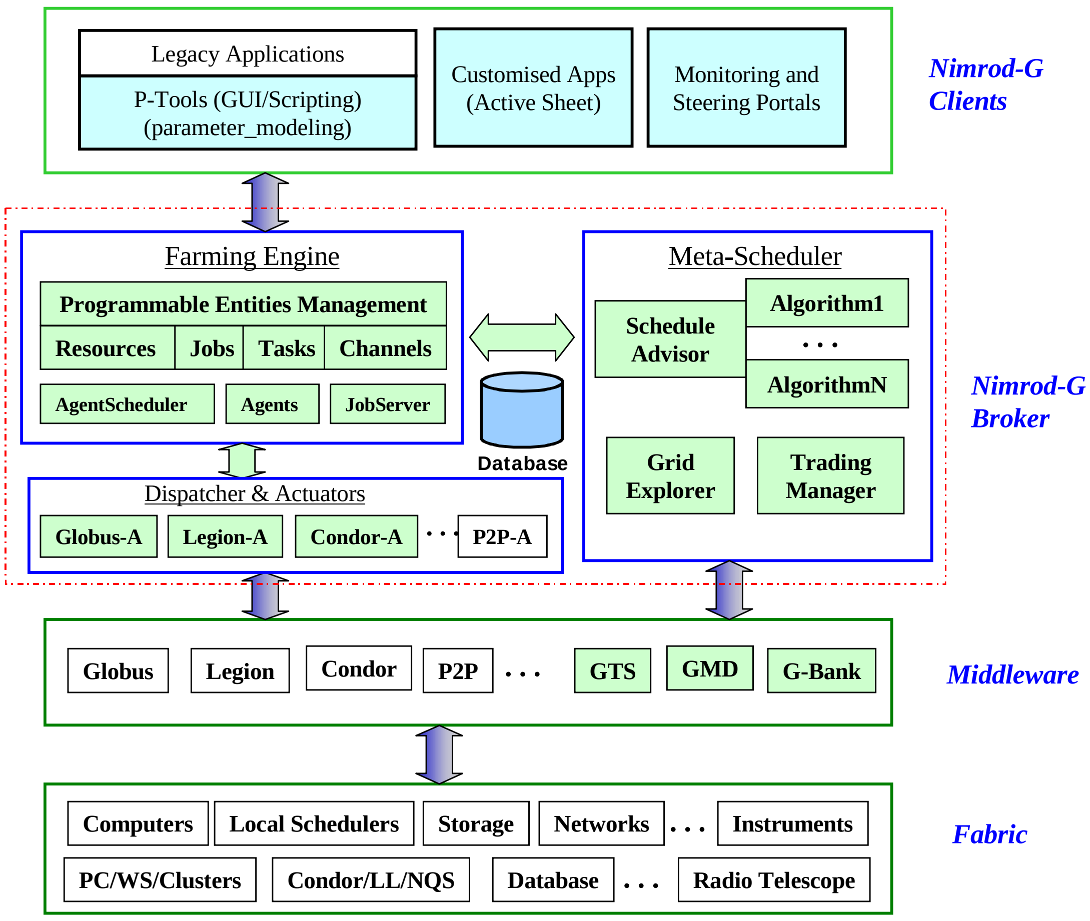


Figure 4.2: A layered architecture of Nimrod-G system.

The key components of Nimrod-G resource broker consist of:

- Nimrod-G Clients, which can be:
  - Tools for creating parameter sweep applications.
  - Steering and control monitors, and
  - Customised end user applications (e.g., ActiveSheets [20]).
- The Nimrod-G Resource Broker, that consists of:
  - A Task Farming Engine (TFE),
  - A Scheduler that performs resource discovery, trading, and scheduling,
  - A Dispatcher and Actuator, and
  - 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.

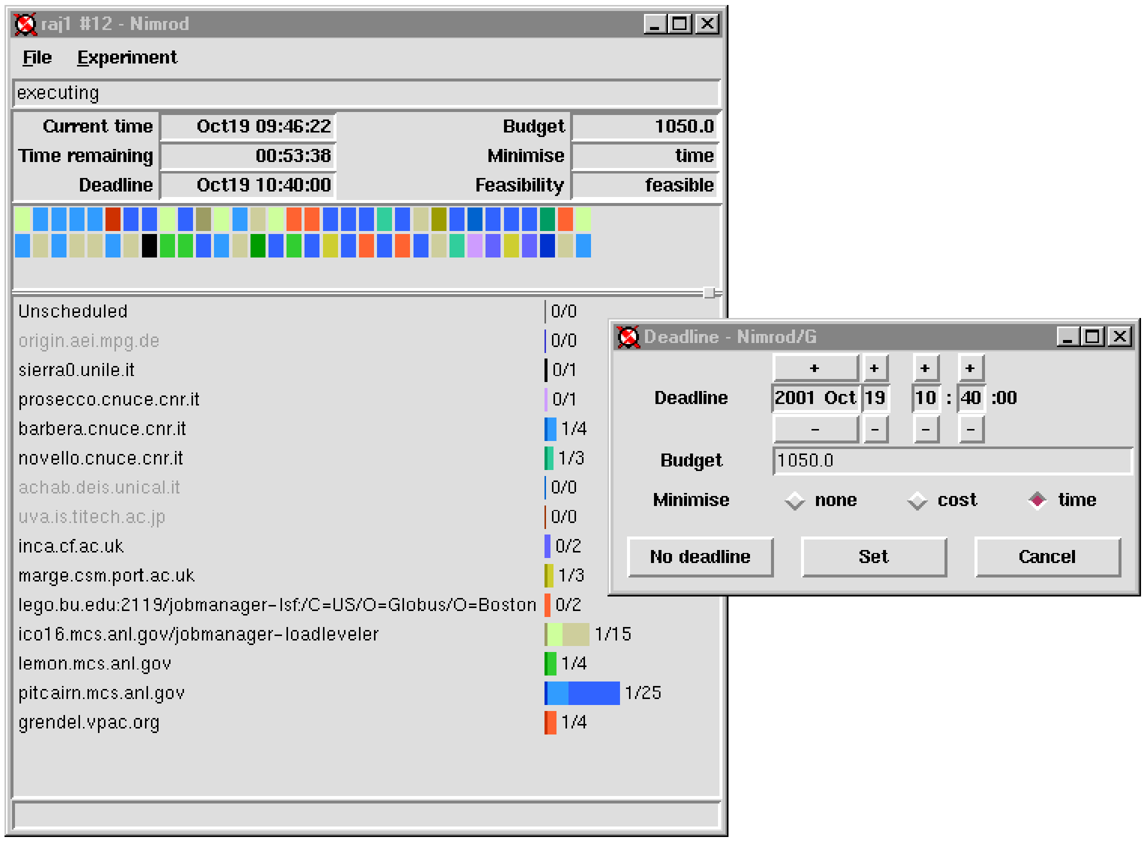
4.2.4 Nimrod-G Clients

Tools for Creating Parameter Sweep Applications

Nimrod supports GUI tools and declarative programming language that assist in creation of parameter sweep applications [21]. They allow the user to: a) parameterise input files, b) prepare a plan file containing the commands that define parameters and their values, c) generate a run file, which converts the generic plan file to a detailed list of jobs, and d) control and monitor execution of the jobs. The application execution environment handles online creation of input files and command line arguments through parameter substitution.

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 4.3. Another feature of the Nimrod-G client is that it is possible to run multiple instances of the same client at different locations. That means the experiment can be started on one machine, monitored on another machine by the same or 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.



executing

Current time	Oct19 09:46:22	Budget	1050.0
Time remaining	00:53:38	Minimise	time
Deadline	Oct19 10:40:00	Feasibility	feasible

Unscheduled	0/0
origin.aei.mpg.de	0/0
sierra0.unile.it	0/1
prosecco.cnuce.cnr.it	0/1
barbera.cnuce.cnr.it	1/4
novello.cnuce.cnr.it	1/3
achab.deis.unical.it	0/0
uva.is.titech.ac.jp	0/0
inca.cf.ac.uk	0/2
marge.csm.port.ac.uk	1/3
lego.bu.edu:2119/jobmanager-lsf:/C=US/O=Globus/O=Boston	0/2
ico16.mcs.anl.gov/jobmanager-loadleveler	1/15
lemon.mcs.anl.gov	1/4
pitcairn.mcs.anl.gov	1/25
grendel.vpac.org	1/4

Deadline 2001 Oct 19 10 : 40 :00

Budget 1050.0

Minimise none cost [✓ time]

Figure 4.3: A Snapshot of Nimrod-G Execution Monitoring and Steering Client.

Customised End User Applications

Specialized applications can be developed to create jobs at runtime and add jobs to the Nimrod-G Engine for processing on the Grid. These applications can use the Nimrod-G job management services (APIs and protocols described in [134]) for adding and managing jobs. One such application is ActiveSheets [20], an extended Microsoft Excel spreadsheet that submits cell functions as jobs to the Nimrod-G broker for parallel execution on the Grid (see Figure 4.4). Another example is the Nimrod/O system, a tool that uses non-linear optimization algorithms to facilitate automatic optimal design [16]. This tool has been used on a variety of case studies, including antenna design, smog modeling, durability optimization, aerofoil design, and computational fluid dynamics [17].

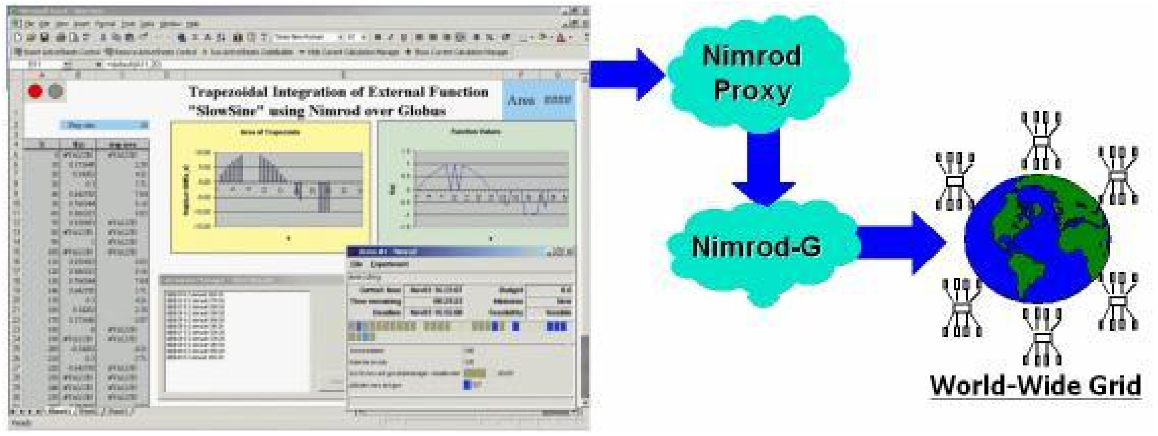

Trapezoidal Integration of External Function "SlowSine" using Nimrod over Globus
Nimrod Proxy
Nimrod-G
World-Wide Grid

Figure 4.4: Active Sheet – Spreadsheet processing on the Grid using the Nimrod-G broker.

4.2.5 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 sub-modules 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 4.5. 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 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*.

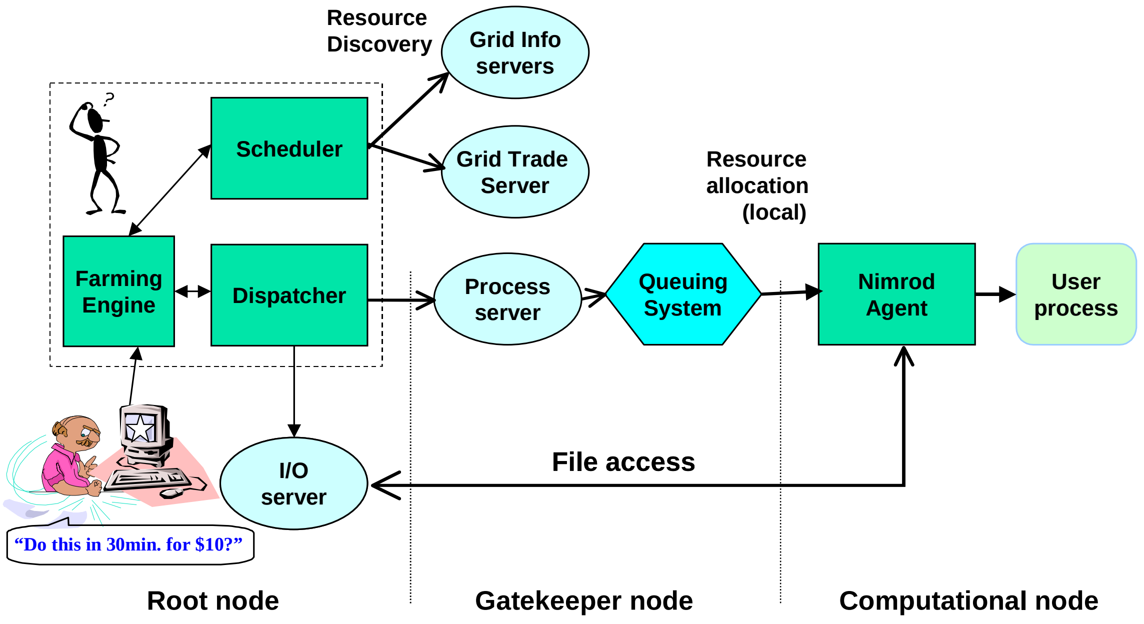


Figure 4.5: The Flow actions in the Nimrod-G runtime environment.

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 parameterised application jobs. It takes care of 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 described in [134]. 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 customised clients or problem solving environments (e.g., ActiveSheets [20]) 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.

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 incorporates three different algorithms (discussed in section 4.4) for deadline and budget constrained scheduling [105].

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 based on 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 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.

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.

4.3 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 application 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 like 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 Time-scale (like high @ daytime and low @ 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 optimisation preference into consideration. In the next section, we discuss deadline and budget constrained (DBC) algorithms that we developed for scheduling parameter sweep applications on globally distributed Grid resources.

4.4 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 data sets. A range of scenarios and parameters to be explored are applied to the program input values to generate different data sets. The programming and execution model of such applications resemble the SPMD (Single Program Multiple Data) 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 quality of service 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
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 post-processing, if required.

The high-level steps for scheduling with deadline and budget constraints are shown in Figure 4.6.

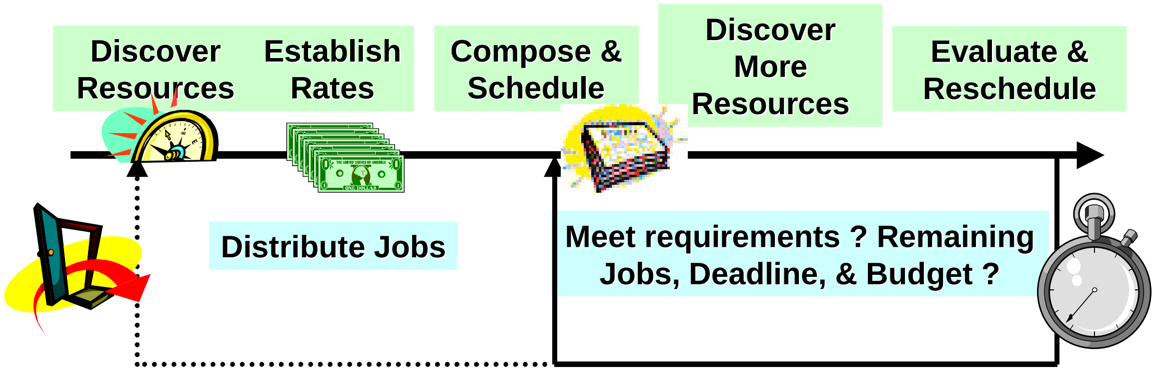


Figure 4.6: High level steps for adaptive scheduling used in the Nimrod-G broker.

The scheduling and orchestration of the execution of parameter sweep applications on world-wide distributed computers appears simple, but complexity arises when users place QoS constraints like deadline (execution completion time) and computation cost (budget) limitations. Such a guarantee of service is hard to provide in a Grid environment since its resources are shared, heterogeneous, distributed in nature, and owned by different organisations 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 Optimisation, within time and budget constraints,
- Time Optimisation, within time and budget constraints,
- Conservative Time Optimisation, within time and budget constraints.

The role of deadline and budget constraints in scheduling and objectives of different scheduling algorithms are illustrated in Table 4.2.

Table 4.2: Deadline and Budget constrained scheduling algorithms and objectives.

Scheduling Algorithm Strategies	***Execution Time (not beyond the deadline)***	***Execution Cost (not beyond the budget)***
Cost Optimisation	Limited by deadline	Minimise
Time Optimisation	Minimise	Limited by budget
Conservative Time Optimisation	Limited by deadline	Limited by budget

We have developed another new algorithm, called cost-time optimisation scheduling, which extends the first two (cost and time optimisation) scheduling algorithms. This new algorithm and the performance evaluation results are discussed in Chapter 6, *Scheduling Simulations*.

The *Time Optimisation 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 the above steps until all jobs are assigned.

The *Cost Optimisation 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 Optimisation* 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 by 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 dearer resources, repeat all steps (with a recalculated budget per job) until all jobs are assigned.

Note that the implementations of all the above algorithms contain extra steps for dealing with the initial start-up (when the average completion times are unknown), and for when all jobs cannot be assigned to resources (infeasible schedules). Detailed steps of the above scheduling heuristics are described in Chapter 6: "Scheduling Simulations".

4.5 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 [59]. 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 [134] 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 [68] and Web services [120]. To achieve this, it is only necessary to implement 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 [49] and Legion [121] 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 [79] resource management system. The use of various Grid and commodity technologies in implementing Nimrod-G components and functionality is presented in Table 4.3.

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 information, starting from submission to completion, is 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 the management of the TFE database and its interaction with other components.

Table 4.3: The Nimrod-G resource broker modules functionality and the 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. It 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 (described in [134]) for job management. Nimrod-G clients or problem solving environments can add, remove, and enquire about job status. They can set user requirements such as deadline and budget; start and stop application execution both at job level and the entire application level.
Scheduling Interface	The Nimrod-G broker supports protocols and interfaces (described in [134]) for mapping jobs to resources. The schedulers can interact with the 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).
Resource Discovery	Resource discovery involves discovering appropriate resources and their properties that match with 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	The market-driven resource trading is performed using GRACE trading services. The 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	The Nimrod-G scheduler performs the user-level resource capability measurement and load profiling by measuring and establishing the job consumption rate.
Scheduling Algorithms	Deadline and budget-based constrained (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.

4.6 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 behaviour 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 chapters).

For this simulation, we created experiments containing 100 jobs, each with a 90 second running time, giving a total computation time of 9000 seconds. 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-second. The optimal deadline for this experiment is achieved when each queue runs 10 jobs in sequence, giving a running time of 900 seconds for the 100 jobs.

We selected three deadlines: 990 seconds (the optimal deadline plus 10%), 1980 seconds (990 x 2), and 2970 seconds (990 x 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 252000 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. A budget of 171000 G$ is the budget required to execute 10 jobs on each of the queues. Finally, the lowest budget of 126000 G$ is the budget required to execute 20 jobs on each of the 5 cheapest queues. Note that for this value, the deadline of 990 seconds is infeasible, and the deadline of 1980 seconds is the optimal deadline plus 10%.

Table 4.4 shows a summary of results for each combination of scheduling algorithm, deadline and budget, and the resulting percentage of completed jobs, the total running time, 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.

Table 4.4: Behaviour of scheduling algorithms for various scenarios on the Grid.

Algorithm	Deadline	Budget	Completed	Time	Cost (G$)	Remarks
Cost Optimisation	990	126000	85%	946	125820	Infeasible
	990	171000	84%	942	139500	Hard
	990	252000	94%	928	156420	Hard
	1980	126000	97%	1927	124740	Hard
	1980	171000	99%	1918	128520	
	1980	252000	98%	1931	127620	
	2970	126000	98%	2931	116820	
	2970	171000	98%	2925	116820	
	2970	252000	100%	2918	118800	
Time Optimisation	990	126000	36%	955	50040	Infeasible
	990	171000	100%	913	171000	Hard

	990	252000	100%	930	171000	Hard
	1980	126000	80%	1968	101340	Hard
	1980	171000	100%	909	171000	
	1980	252000	100%	949	171000	
	2970	126000	100%	2193	126000	
	2970	171000	100%	928	171000	
	2970	252000	100%	922	171000	
Conservative Time Optimisation	990	126000	78%	919	120060	Infeasible
	990	171000	99%	930	168480	Hard
	990	252000	100%	941	171000	Hard
	1980	126000	97%	1902	125100	Hard
	1980	171000	100%	1376	160740	
	1980	252000	100%	908	171000	
	2970	126000	99%	2928	125100	
	2970	171000	100%	1320	161460	
	2970	252000	100%	952	171000	

The behaviour of the queues is analysed by examining the usage of the queues over the period of the experiment. For the Cost Optimisation algorithm, Figure 4.7 shows the node usage for a deadline of 1980 seconds. After an initial spike, during which the scheduler gathers information about the queues, the scheduler calculates that it needs to use the 4-5 cheapest queues only in order to satisfy the deadline. (Actually, it requires exactly 5, 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 minimising 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 126000 units is only enough to complete the experiment if the cheapest 5 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 this is mainly because, in seeking to minimise 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.

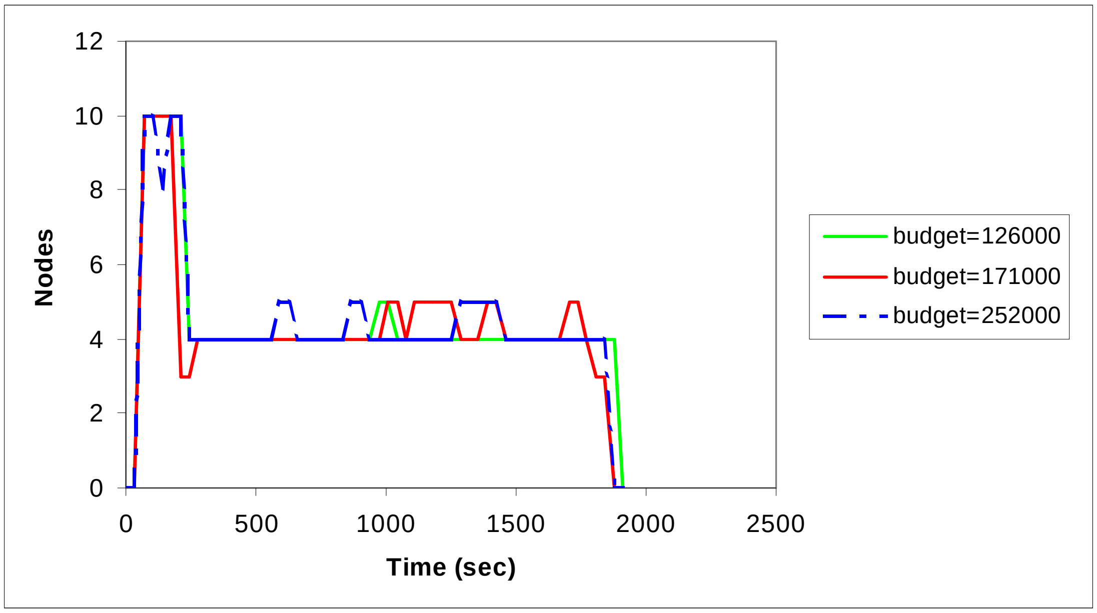


Figure 4.7: DBC Cost-optimisation scheduling algorithm behaviour for various budgets.

The equivalent graph for the Time Optimisation algorithm is shown in Figure 4.8. 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 Optimisation algorithm shown in Figure 4.9, we see a lot more variation in the schedules chosen for different budgets. The schedule with a very large budget is equivalent to the Time Optimisation algorithm. The schedule with the low budget is almost the same as the Cost Optimisation algorithm.

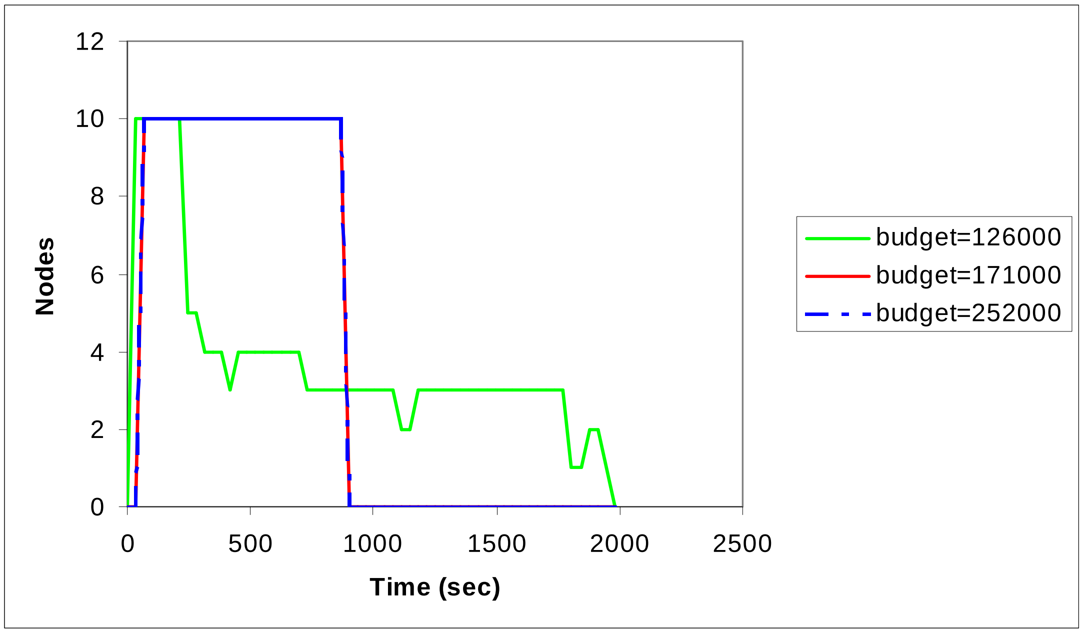


Figure 4.8: Time optimisation scheduling algorithm behaviour for various budgets.

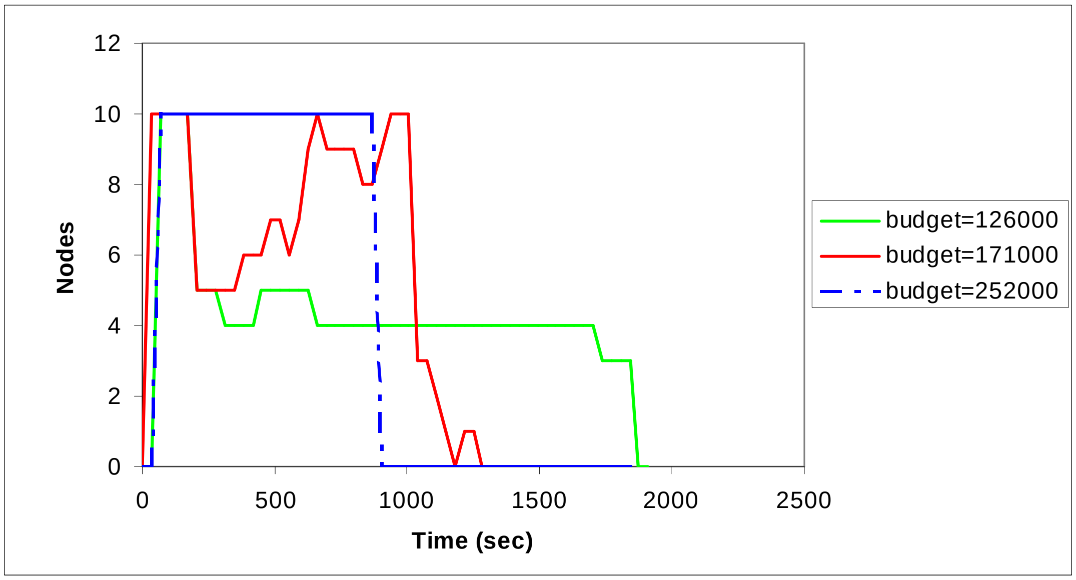


Figure 4.9: Conservative time optimisation scheduling algorithm behavior for different budgets.

4.7 Scheduling Experiments on the World-Wide 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 World Wide Grid (WWG) [111] testbed during each experiment. They can be categorised into the following scenarios:

- Cost optimisation scheduling during Australian peak and off-peak times,
- Cost and time optimisation scheduling using cheap local and expensive remote resources, and
- Large scale scheduling using cost and time optimisation algorithms.

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

4.7.1 The World-Wide Grid (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 4.10 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 thesis 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.

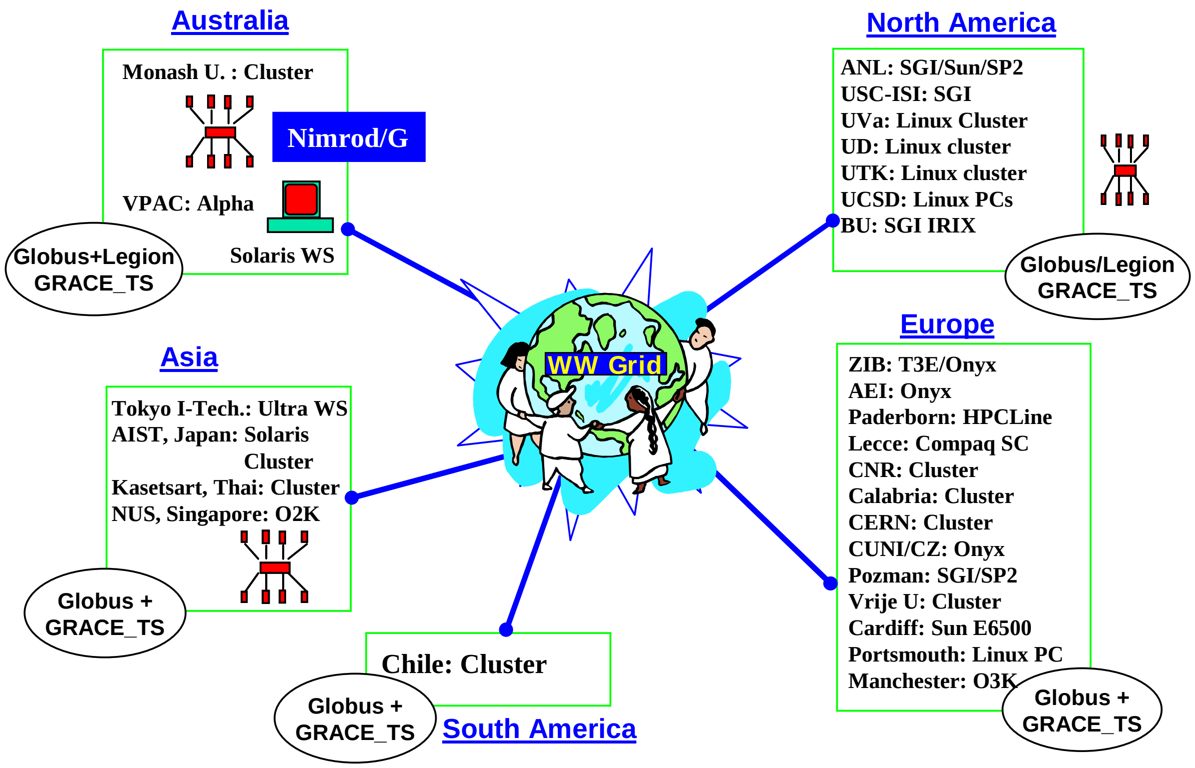


Figure 4.10: The World Wide Grid (WWG) testbed.

The WWG testbed contains numerous computers with different architecture, capability, and configuration. They include PCs, workstations, SMPs, clusters, and vector supercomputers running operating systems such as Linux, Sun Solaris, IBM AIX, SGI IRIX, and Compaq Tru64. Further, the systems use a variety of job management systems such as OS-Fork, NQS, Condor, RMS, PBS, and LSF.

These system characteristics can be identified by accessing the GIS (Grid Information Service) 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 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 paramterisation and task specification, the list of resources that can possibly be utilised, and their QoS requirements such as the deadline, budget, and optimisation 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 modelling 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.

4.7.2 Cost Optimisation Scheduling – Australian Peak and Off-peak Times

In a competitive commodity-market economy, the resources are priced differently at different times based on 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 World-Wide Grid resources, during Australian peak and off-peak hours.

WWG Computational Resources

The World-Wide Grid testbed resources selected for use in this experiment and their properties is shown in Table 4.5. 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 US. The access price is expressed in Grid units (G$) per CPU second.

Table 4.5: World-Wide Grid testbed resources used in the experiment. Price is G$ per CPU sec.

Resource Type & Size (No. of nodes)	Organization & Location	Grid Services and Fabric	Price @ AU peak time	Price @ 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

We selected 5 resources (see Table 4.5) 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 maintained on each of the resources by their owners. The resource cost database contains access cost (price) that they 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. The resource broker negotiates with trading servers for establishing access price using the resource trading services provided by the GRACE infrastructure.

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 degrees. The second parameter `time_base_value` indicates the expected calculation complexity in minutes plus 0 to 60 seconds positive deviation. That means the program *calc* is expected to run for anywhere between 5 to 6 minutes on resources with some variation depending on resource capability. A plan file modelling this application as a parameter sweep application using the Nimrod-G parameter specification language is shown in Figure 4.11. 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 of 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.

```
#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 4.11: Nimrod-G parameter sweep processing specification.

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 *minimise the cost*, within *one-hour deadline*. This requirement instructs the Nimrod-G broker to use "Cost-Optimization Scheduling" algorithm in scheduling jobs for processing on the Grid.

The number of jobs in execution/queued on resources during the Australian peak and off-peak time scheduling experimentations is shown in Figure 4.12 and Figure 4.13 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 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 Figure 4.13). 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.

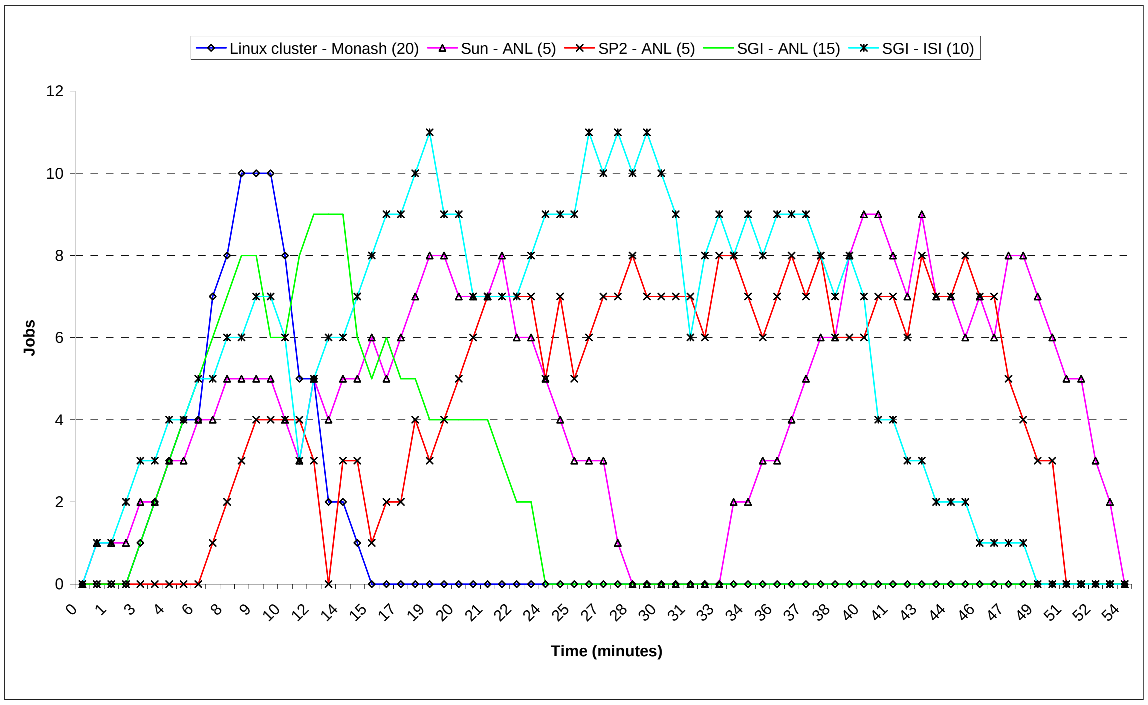


Figure 4.12: Computational scheduling during Australian peak time (US off-peak time).

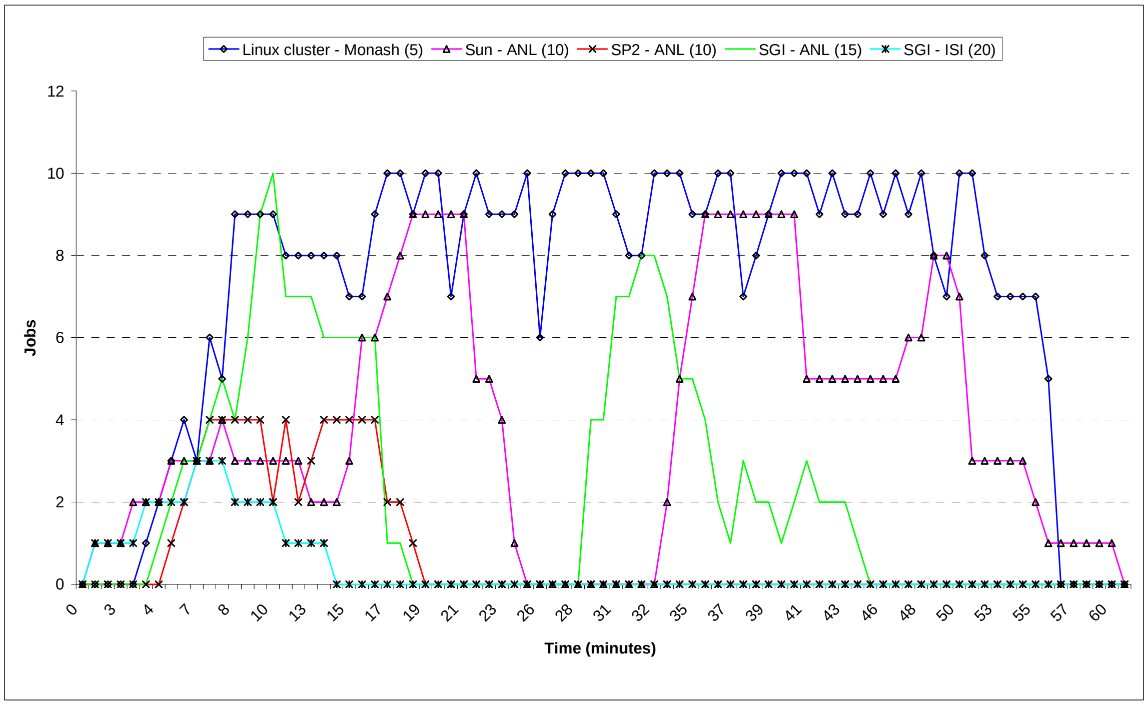


Figure 4.13: Computational scheduling during Australian off-peak time (US peak time).

When the scheduling algorithm tries to minimize the cost, the total cost Australian peak time experiment is 471205 G$ and the off-peak time is 427155 G$. The result is that costs were quite low in both cases. An experiment using all resources, without the cost optimization algorithm during the Australian peak, costs 686960 G$ for the same workload. The cost difference indicates a saving in computational cost and it is certainly a successful measure of our budget and 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 4.14. 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 tried to use as many resources as possible to ensure that it can meet deadline. After calibration phase, 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 it ensures that the cost of computation is within a given budget.

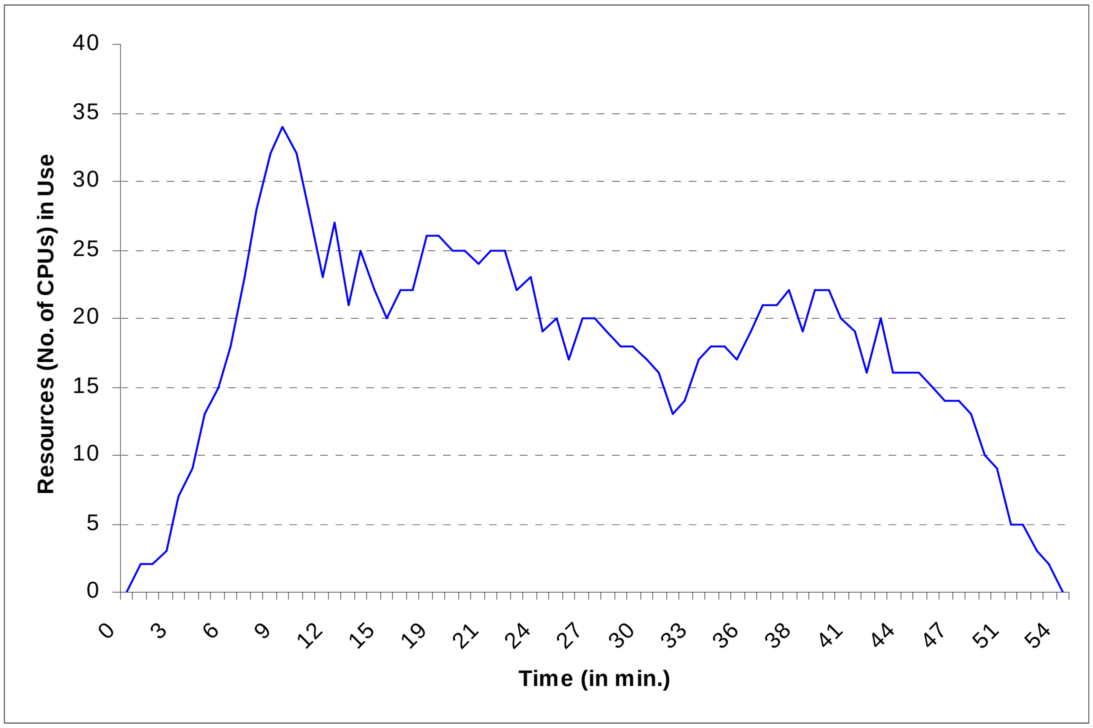


Figure 4.14: Number of resources in use during Australian peak time scheduling experiment.

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 4.15. It can be observed that the pattern of variation of cost during 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 that the scheduler selected were from off-peak time zone (i.e., US was in off-peak time when Australia was in peak hours) as they were cheaper. Another reason is that the number of resources used in these experiments contains more US resources compared to Australian resources.

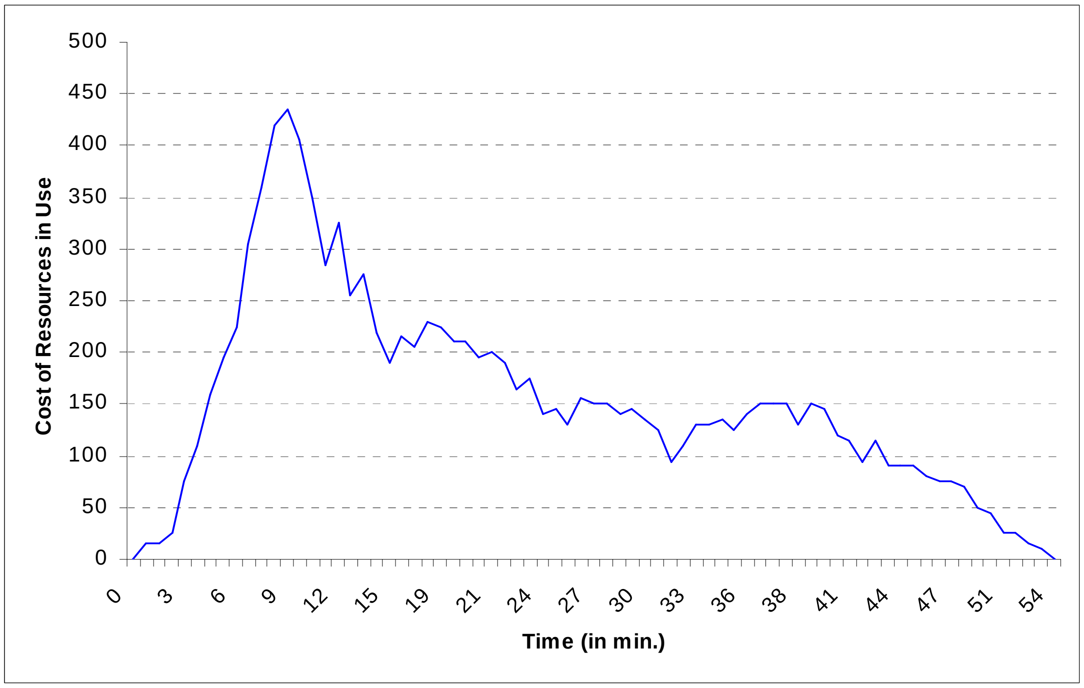


Figure 4.15: Cost of resources in use at Australian peak time scheduling experiment.

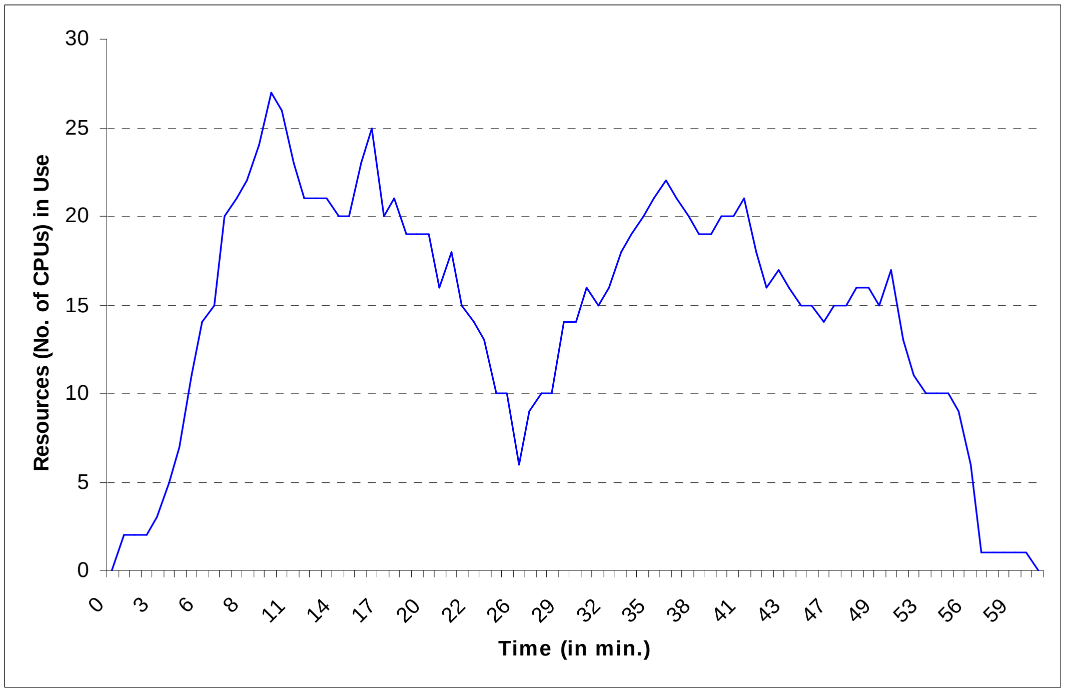


Figure 4.16: Number of resources in use at Australian off-peak time scheduling experiment.

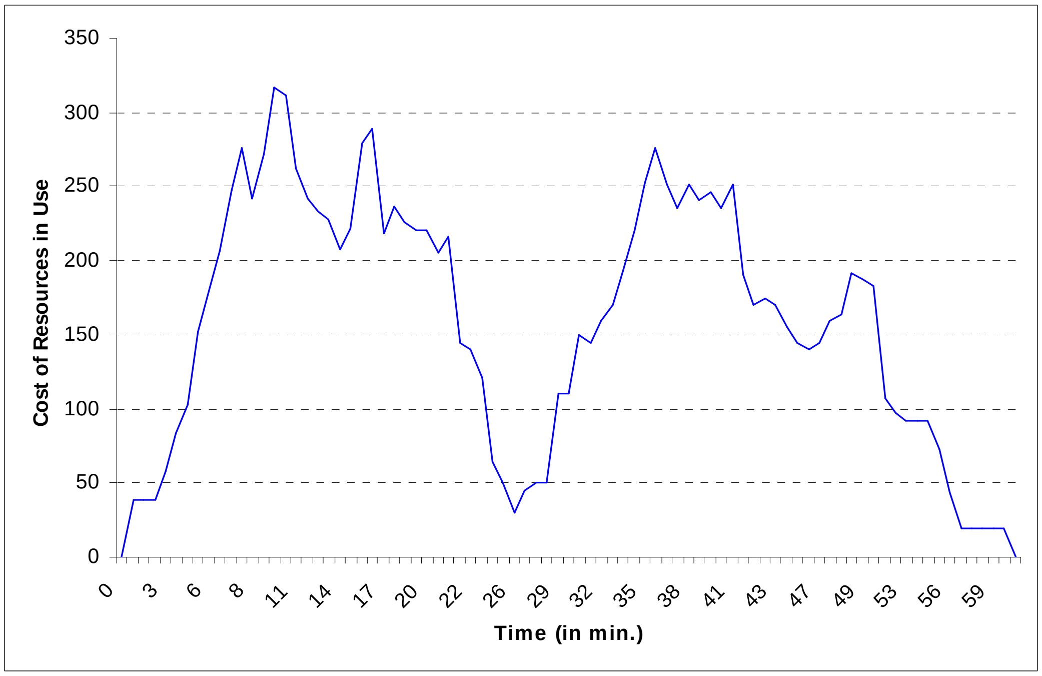


Figure 4.17: Cost of resources in use at Australian off-peak time scheduling experiment.

Similar behavior did not occur in scheduling experiments conducted during Australian off-peak time (see Figure 4.16 and Figure 4.17). The variation pattern of total number of resources in use and their total cost is similar due to the fact that the larger numbers of US resources were available cheaply. Although the scheduler has used Australian resources throughout the experiment (see Figure 4.13), the scheduler had to depend on US resources to ensure that the deadline is met even if resources were expensive.

4.7.3 Cost and Time Optimisation 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 (same as used in the previous scheduling experiment) containing 165 CPU-intensive jobs, each running approximately 5 minutes duration.

We have set the deadline of 2 hours (120 minutes) and budget of 396000 (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.
- 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 in [103] 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 4.6 shows resources details such as architecture, location, and access price along with type of Grid middleware systems used in making 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 using the GRACE resource trading protocols (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.

Table 4.6: The WWG testbed resources used in scheduling experiments, job execution and costing.

Resource Type & Size (No. of nodes)	**Organization & Location**	**Grid Services and Fabric**	**Price (G$ per CPU sec.)**	**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$)		237000	115200
		Time to Complete Experiment (Min.)		70	119

The number of jobs in execution on resources (Y-axis) at different times (X-axis) during the experimentation is shown in Figure 4.18 and Figure 4.19 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 minutes* and spent *237000 G$*. In the second experiment (cost minimization), the broker selected cheap resources as much as possible to minimize the execution cost whilst still trying to meet the deadline (completed in *119 minutes*) and spent *115200 G$*. 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 due to 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.

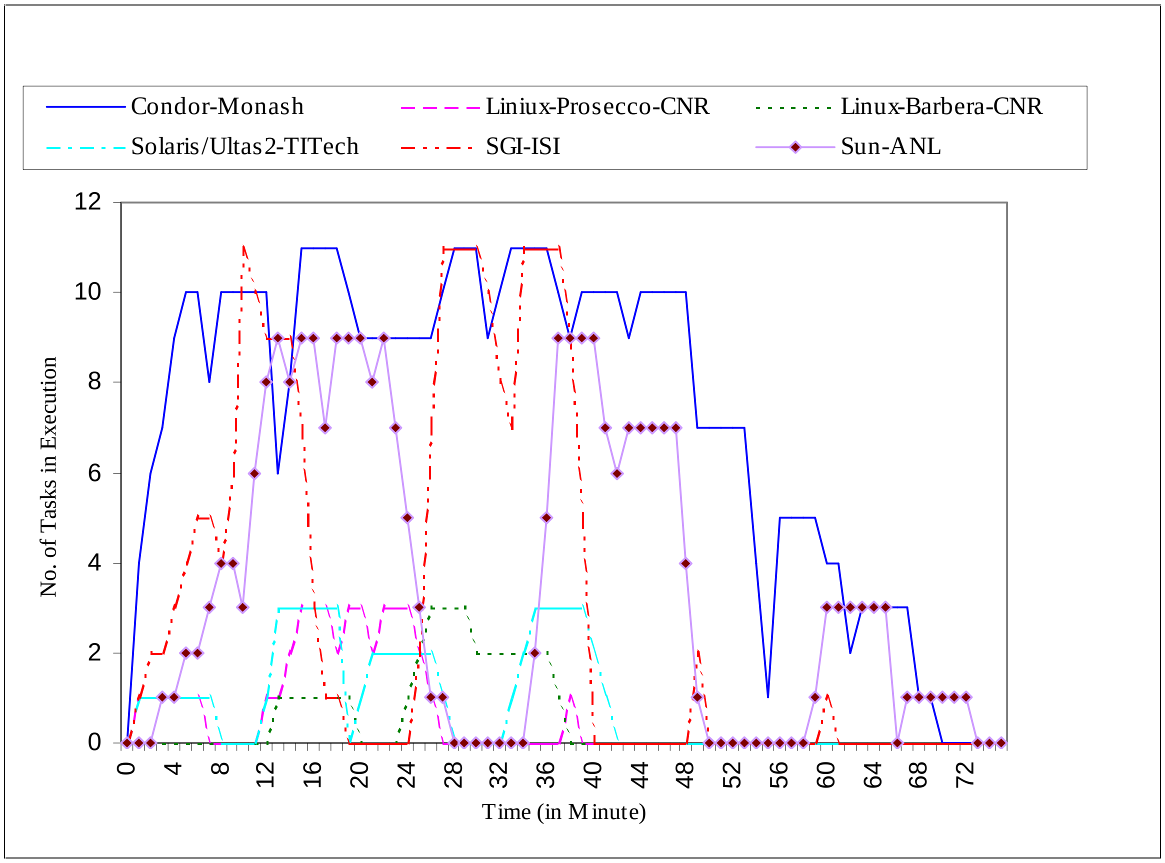


Figure 4.18: Resource selection in deadline and budget constrained time optimization scheduling.

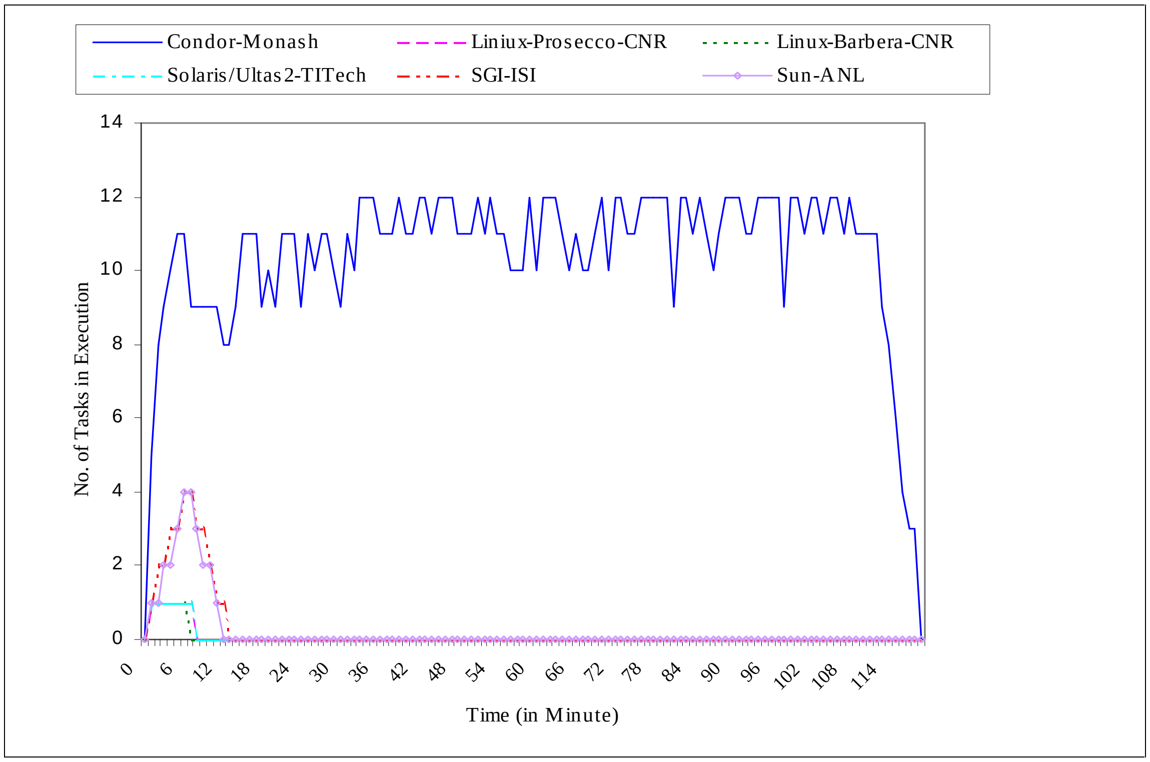


Figure 4.19: Resource selection in deadline and budget constrained cost optimization scheduling.

4.7.4 Large Scale Scheduling with Cost and Time Optimisation

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 angle in degree 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. That means, the program *calc* is modeled to execute for anywhere between 10 to 11 minutes on resources randomly. A plan for modelling this application as a parameter sweep application using the Nimrod-G parameter specification language is shown in Figure 4.20. 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 step 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 executable 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.

```
#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
   #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 4.20: Nimrod-G parameter sweep processing specification.

We have conducted two scheduling experiments for a given deadline of 4 hours and budget of 250000 (G$ or tokens) with different optimization strategies [105]:

- Optimize for Time: This strategy produces results as early as possible, but before a deadline and within a budget limit.
- Optimize for Cost: This strategy produces results by deadline, but reduces cost within a budget limit.

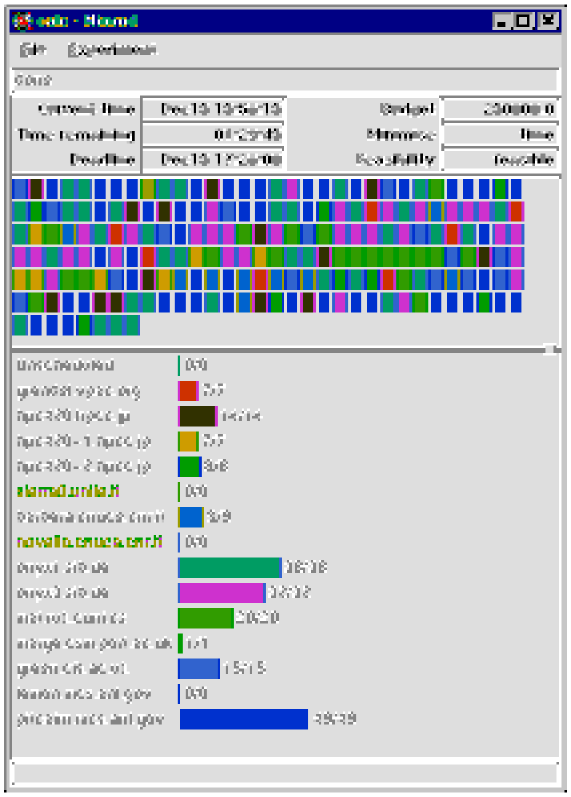
In the case of "optimize for cost" scheduling, the deadline has been extended by 20 minutes after 3 hours of experiment duration to demonstrate that the users can change their QoS requirements at anytime.

In these scheduling experiments, the Nimrod-G resource broker employed the c*ommodity market* model for establishing a service access price. It used Grid resource trading services for establishing 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 for negotiating access to resources and choosing appropriate ones. The access price varies with consumer and 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 does 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.

We have used a subset of resources of the WWG testbed in these scheduling experimentations. Table 4.7 shows resources properties such as architecture, location, and access price, and middleware systems loaded on them. A snapshot of the Nimrod-G monitoring and steering clients taken immediately after the completion of application processing is shown in Figure 4.21 and Figure 4.22. The resources used in both experiments are (time/space) shared resources. They were partially available to us as they were shared and also utilized by other users whose requirements and priorities were different. Similar to the earlier experimentation, the access prices shown in the table are being established dynamically using the GRACE resource trading protocols. The value assigned are artificial, however, they reflect the offering of differentiated services at different costs as in the real-world marketplace.

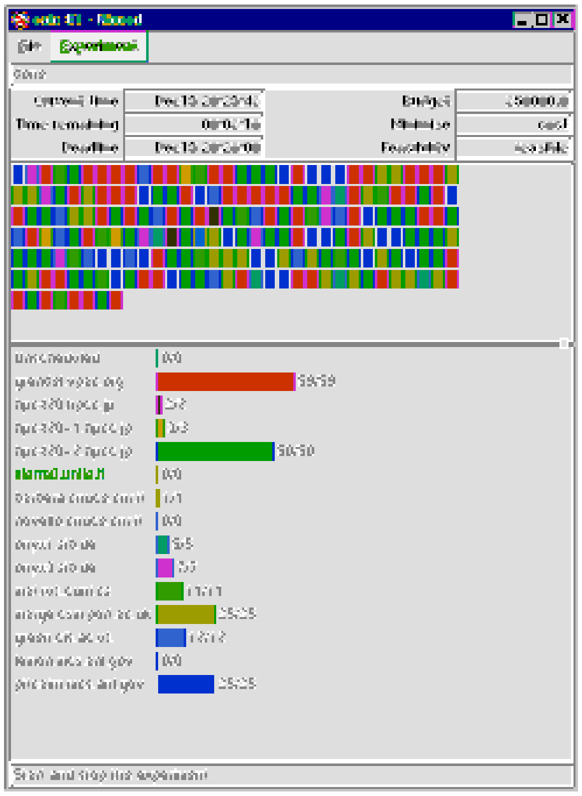
Table 4.7: The WWG testbed resources used in scheduling experiments, job execution and costing.

Organization & Location	Vendor, Resource Type, # CPU, OS, hostname	Grid Services, Fabric, and Role	Price (G$ per CPU sec.)	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, UK	Unknown: Dual CPU PC, Linux, marge.csm.port.ac.uk	Globus, GTS, Fork (Worker node)	1	1	25
University of Manchester, UK	SGI: Onyx3K, 512 node, IRIX, green.cfs.ac.uk	Globus, GTS, NQS, (Worker node)	2	15	12
Argonne National Lab, Chicago, USA	SGI: IRIX lemon.mcs.anl.gov	Globus, GTS, Fork (Worker node)	2	0	0
Argonne National Lab, Chicago, USA	Sun: Ultra –8, Solaris, 8, pitcairn.mcs.anl.gov	Globus, GTS, Fork (Worker node)	1	49	25
		Total execution cost (G$)		199968	141869
		Total execution time (min.)		150	258



Current Time Dec 15 15:58:15
Time remaining 01:29:45
Deadline Dec 15 17:28:00
Budget 230000.0
Minimise time
Feasibility feasible

Figure 4.21: A snapshot of the Nimrod-G monitor during "optimize for time" scheduling experiment.



Current Time Dec 15 20:23:44
Time remaining 00:04:16
Deadline Dec 15 20:28:00
Budget 250000.0
Minimise cost
Feasibility feasible

Figure 4.22: A snapshot of the Nimrod-G monitor during "optimize for cost" scheduling experiment.

DBC Constrained Time Optimization Scheduling

The first experiment, *Optimize for Time* scheduling, we performed on December 15, 2001 at 13.28:00, Australian Eastern Daylight Saving Time (AEDT), with 4 hours deadline and finished on the same day by 15:58:15. A snapshot of the Nimrod-G monitoring and steering client, taken immediately after the completion of this experiment, is shown in Figure 4.21. This experiment took 2½ hours to finish the processing of all jobs using resources available at that time with an expense of 199968 G$. Figure 4.23 shows the number of jobs in execution on different resources and Figure 4.24 shows the total number of jobs in execution on the Grid during the experiment execution period. It can be observed during 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.

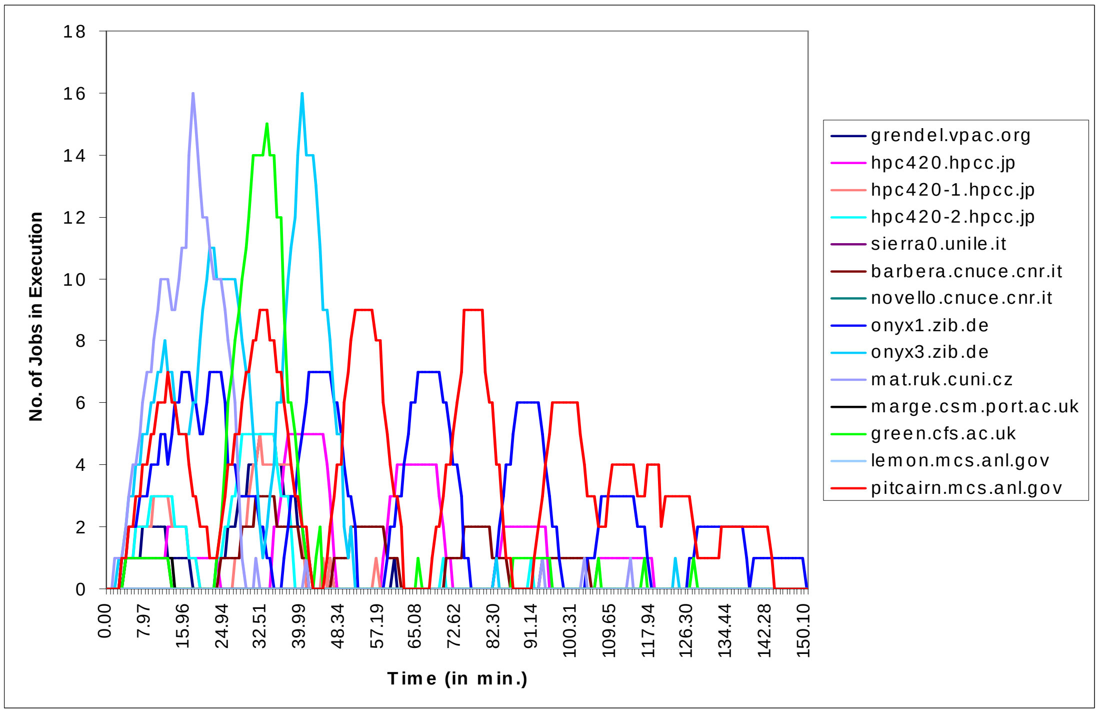


Figure 4.23: No. of jobs in execution on different Grid resources during DBC time optimization scheduling.

Throughout the experiment, two resources, marked in *gray* color in Figure 4.21, 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 4.25 shows the number of jobs processed on different resources selected depending on their cost and availability. Figure 4.26 shows the total number of jobs processed on the Grid. Figure 4.27 shows the corresponding expenses of processing on different resources and Figure 4.28 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 depending on the amount of remaining budget. Another interesting pattern to be observed in Figure 4.25 and Figure 4.27 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 (3G$/sec.), 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 for processing on it.

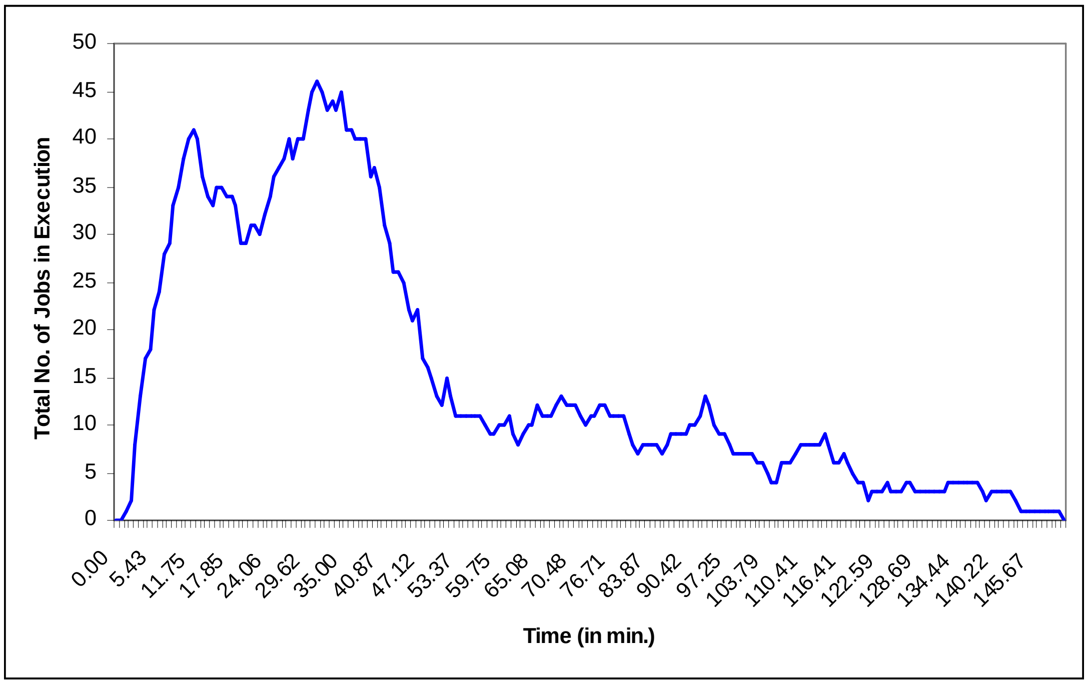


Figure 4.24: Total No. of jobs in execution on Grid during DBC time optimization scheduling.

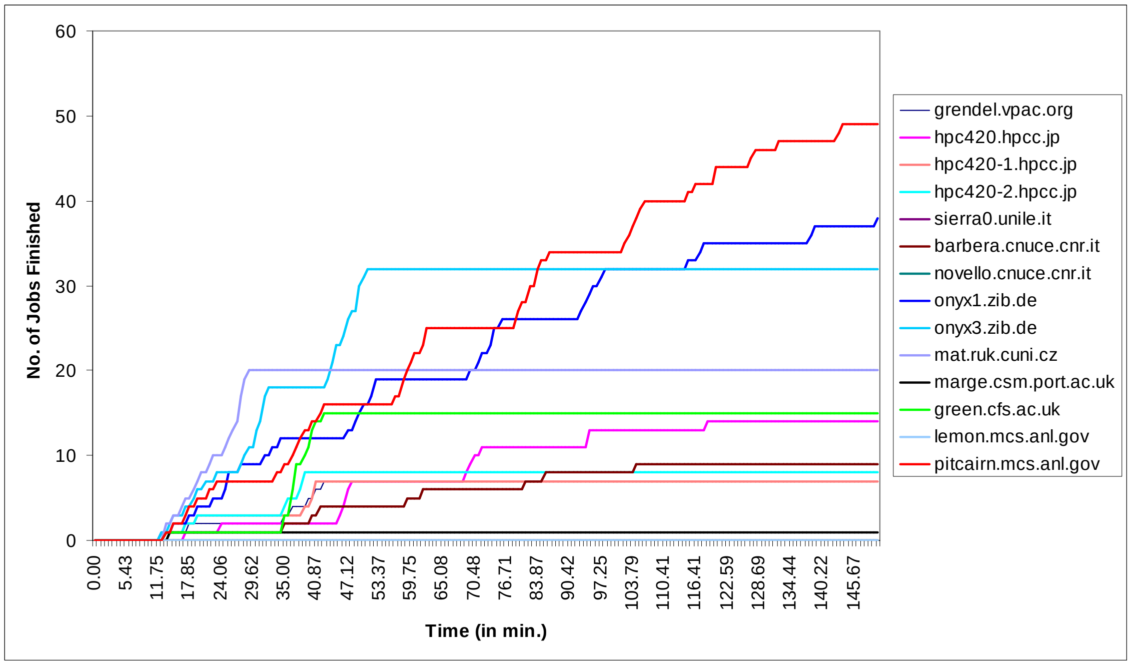


Figure 4.25: No. of jobs processed on different Grid resources during DBC time optimization scheduling.

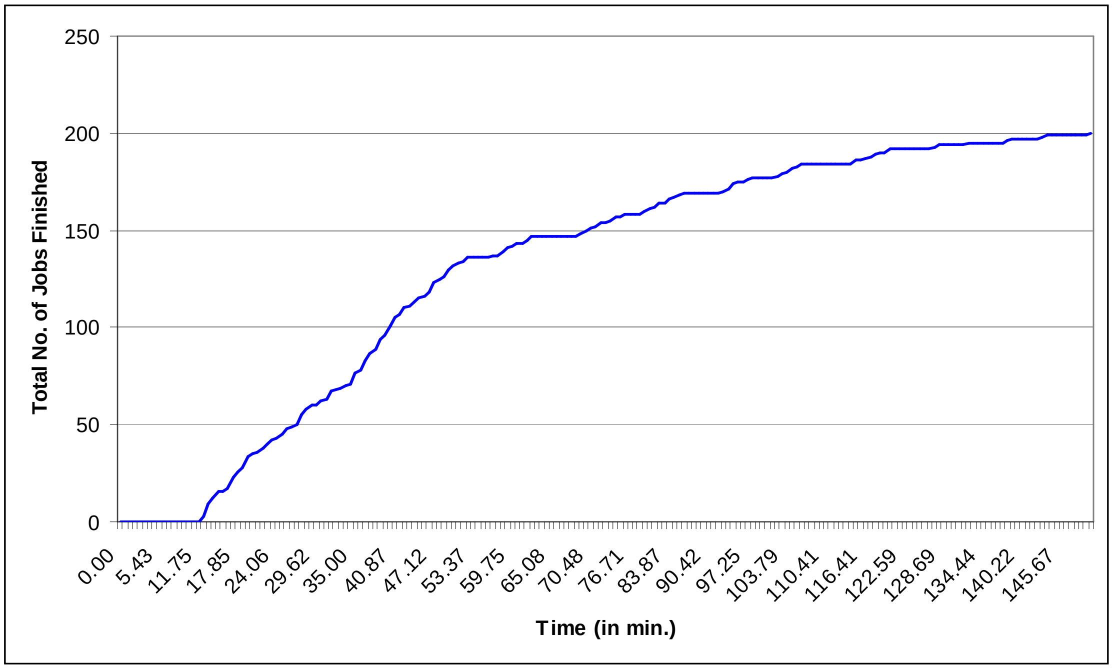


Figure 4.26: Total no. of jobs processed on Grid resources during DBC time optimization scheduling.

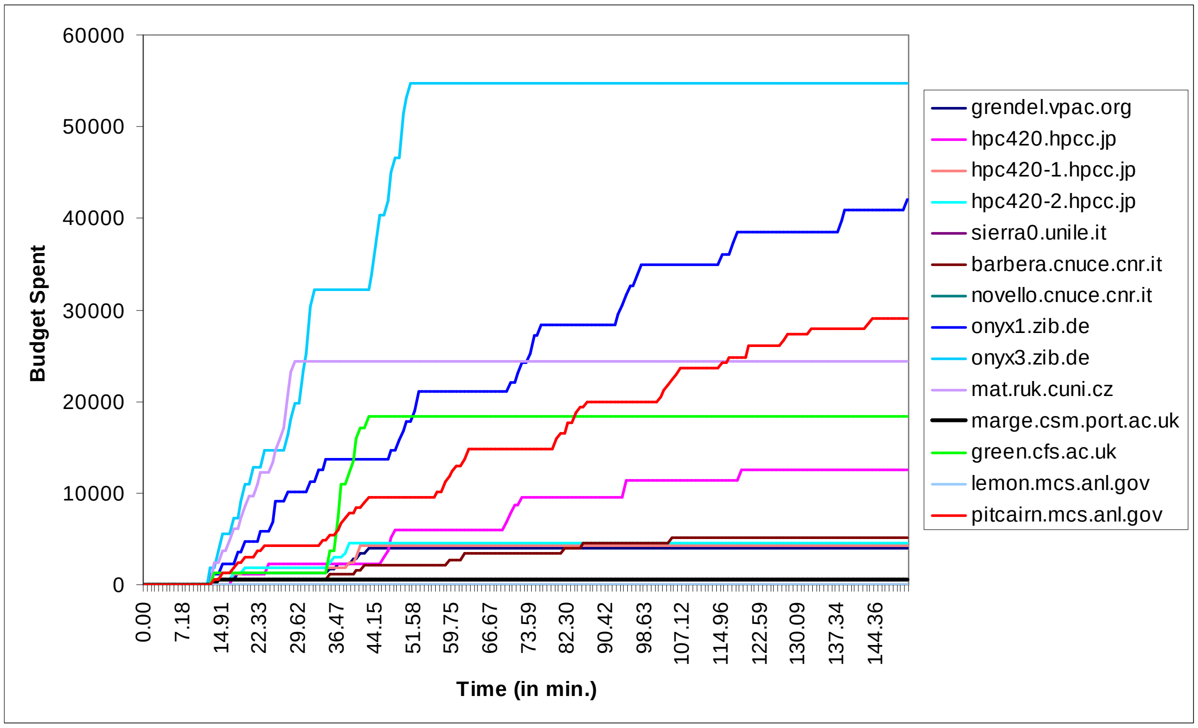


Figure 4.27: The amount spent on different Grid resources during DBC time optimization scheduling.

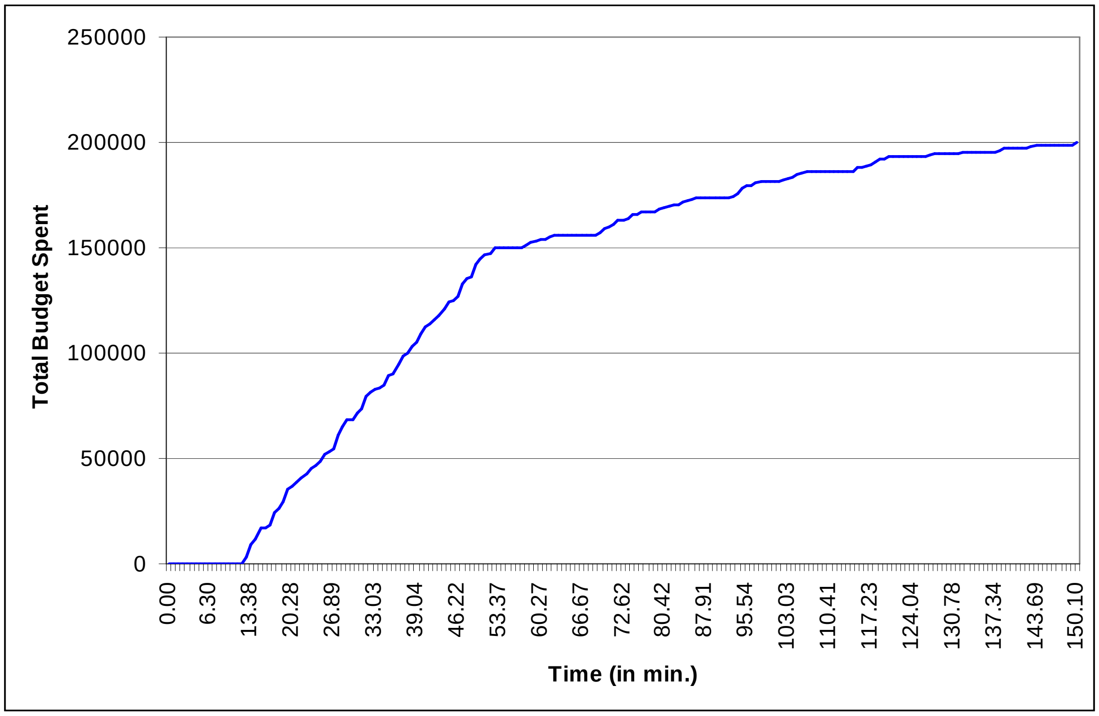


Figure 4.28: The total amount spent on Grid during DBC time optimization scheduling.

DBC Constrained Cost Optimization Scheduling

The second experiment, *Optimize for Cost* scheduling, was performed on December 15, 2001 at 16.10:00, Australian Eastern Daylight Saving Time (AEDT), with 4 hours deadline, which had been extended by 20 minutes after 3 hours of execution time. It finished on the same day by 20:25:52. A snapshot of the Nimrod-G monitoring and steering client, taken immediately after the completion of this experiment, is shown in Figure 4.22. 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 141869 G$. Even though it took more time compared to the first experiment, it saved 58099 G$ from expenses.

Figure 4.29 shows the number of jobs in execution on different resources and Figure 4.30 shows the total number of jobs in execution on the Grid during the experiment execution period. It can be observed 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, marked in *gray* color in Figure 4.22, 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 4.31 shows the number of jobs processed on different resources selected depending on their cost and availability and Figure 4.32 shows the aggregation of jobs processing on the Grid at different times during the experimentation. The graphs (Figure 4.33 and Figure 4.34) show the corresponding amount of budget spent for processing on individual resources and the Grid respectively.

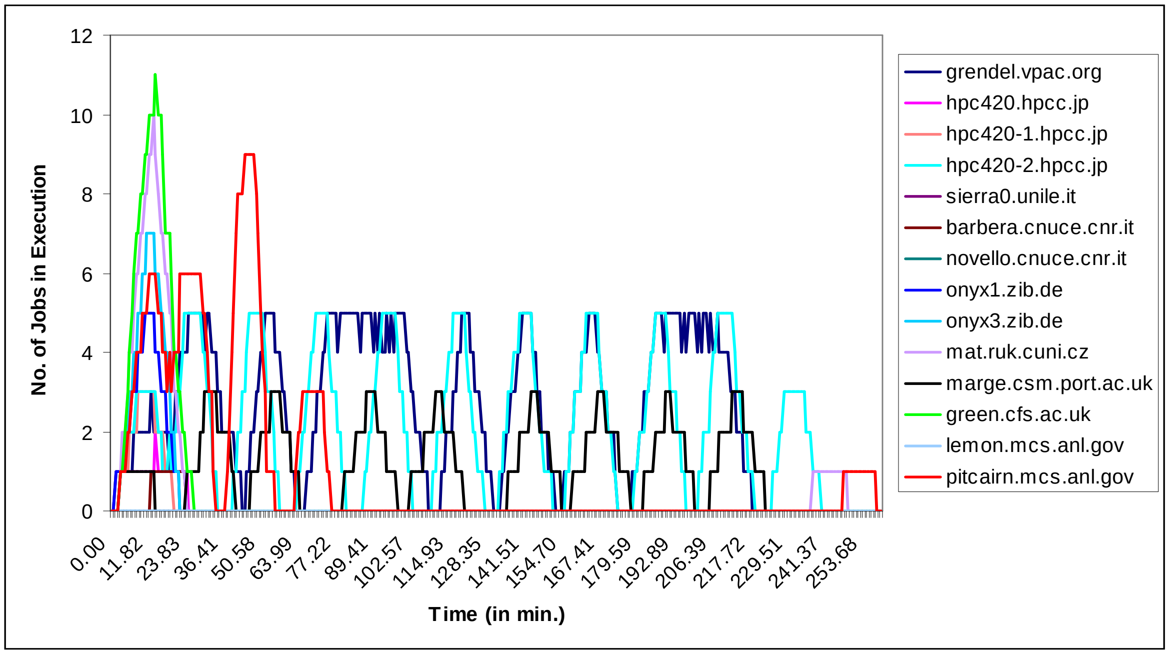


Figure 4.29: No. of jobs in execution on different Grid resources during DBC cost optimization scheduling.

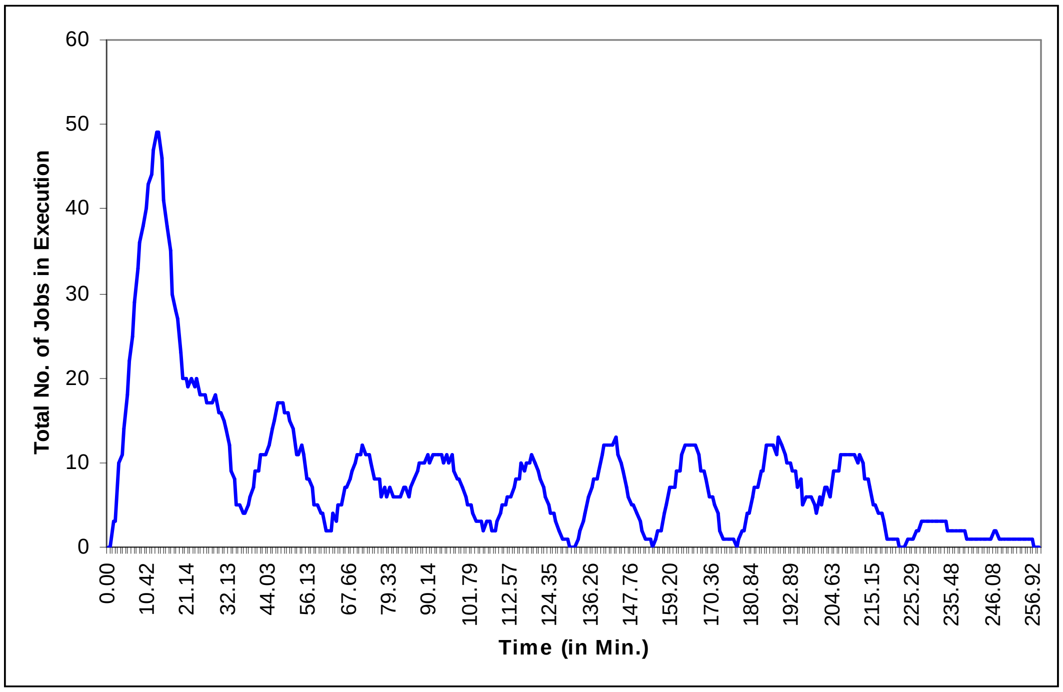


Figure 4.30: Total No. of jobs in execution on Grid during DBC cost optimization scheduling.

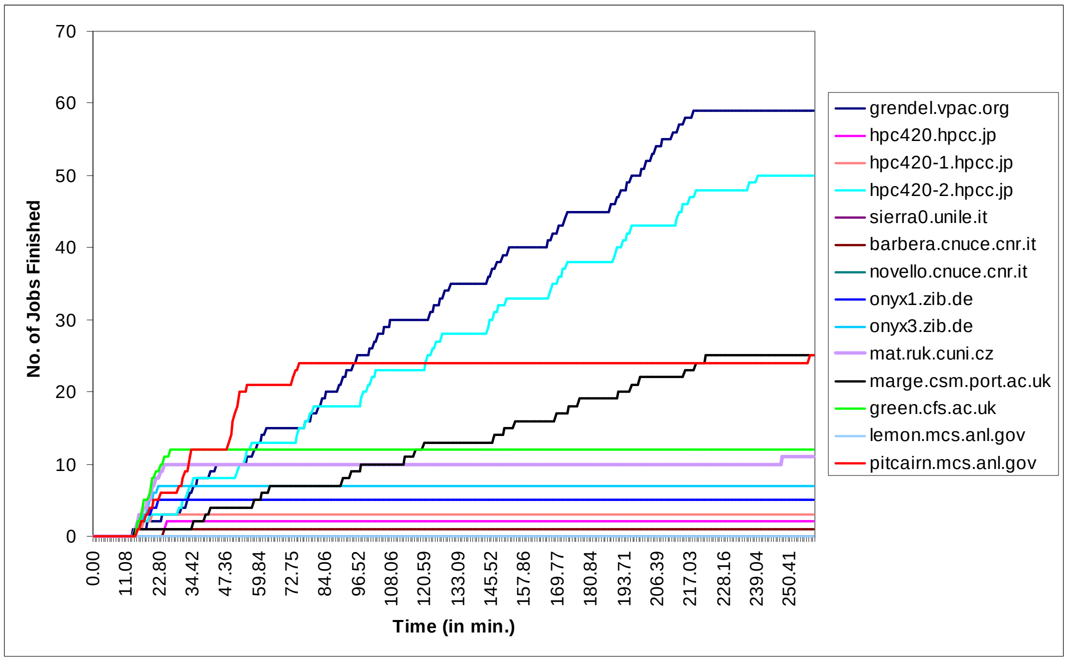


Figure 4.31: No. of jobs processed on different Grid resources during DBC cost optimization scheduling.

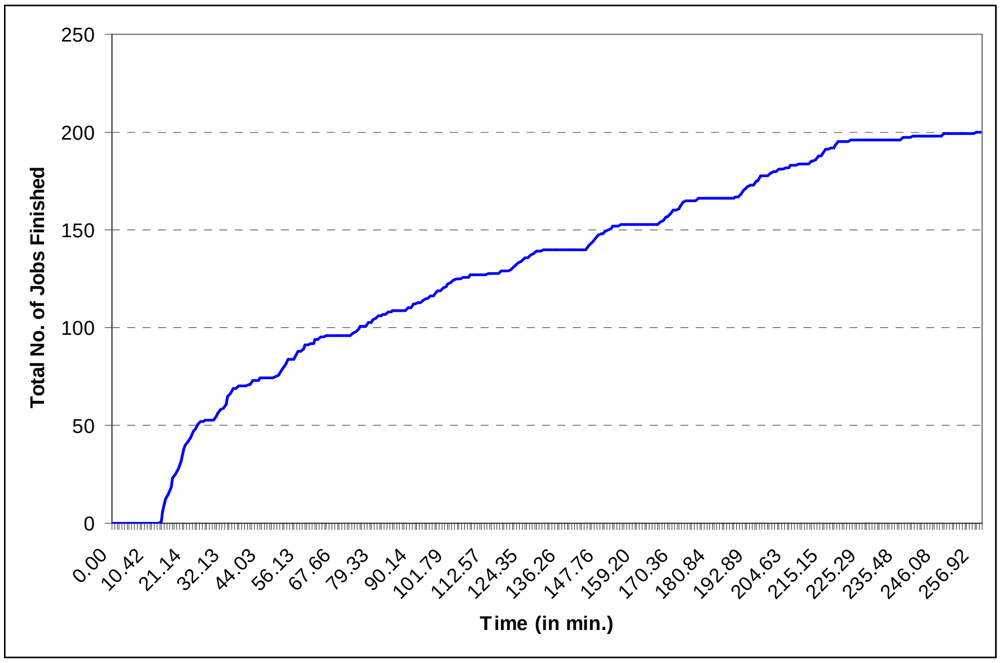


Figure 4.32: Total No. of jobs processed on Grid during DBC cost optimization scheduling.

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 2G$/CPU-sec., to process one job to ensure that the deadline can be met and this was essential as the availability of the cheapest resources had changed from the forecast 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 Figure 4.31 and Figure 4.33) since we had to spend the higher amount for processing on expensive resources.

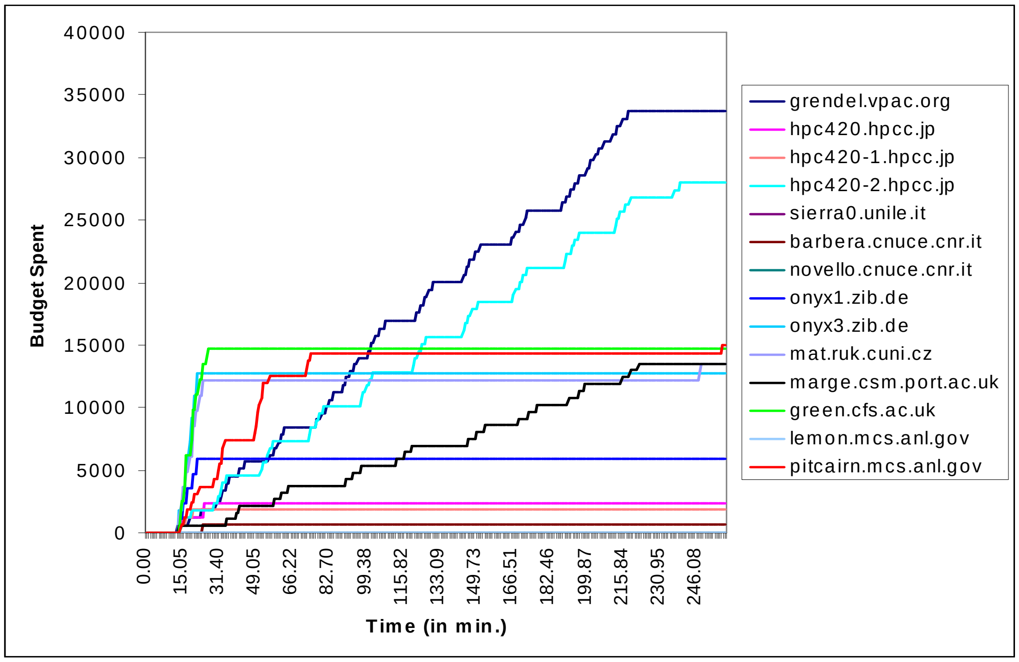


Figure 4.33: The amount spent on different Grid resources during DBC cost optimization scheduling.

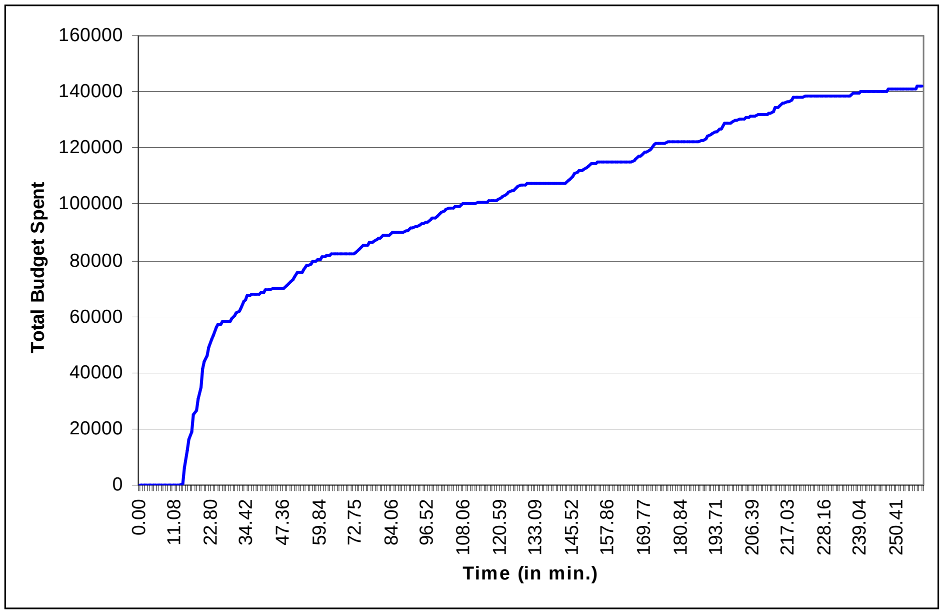


Figure 4.34: The total amount spent on Grid during DBC cost optimization scheduling.

4.8 Summary and Comments

We have discussed the design, development, experimental evaluation of the Nimrod-G Grid resource broker that supports deadline and budget constrained and quality of service requirements-driven application scheduling on world-wide 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 world-wide Grid marketplace.

The Nimrod tools for modeling parametric experiments are quite mature and in production use for cluster computing. A prototype version of Grid enabled tools and Nimrod-G resource broker have been implemented and they are available for download from our project web page. 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 choice and they can indeed trade-off between the deadline and budget depending on their requirements; thus encouraging them to reveal their true requirements to increase the value delivered by the utility.

Although our scheduling experiments on the WWG testbed resources demonstrated the capability of Nimrod-G broker and provided worthwhile insights into our economic scheduling algorithms, we were unable to perform truly comparable and repeatable exhaustive evaluation. For example, with empirical studies in Grid environment, it is impossible to provide answers to questions such as: what if the cost-optimisation scheduling is selected instead of the time optimisation?; what if the deadline is changed from A to B and the budget from X to Y?; what if the parameter range is changed from M to N?; what if resource R failed?; and what if the number of available resources drastically changed?. It is impossible to repeat the scheduling experiment for different requirements with the same resource scenario as in the previous experiment – the availability of Grid resources and load continuously varies with time and it is impossible for an individual user/domain to control activities of other users in different administrative domains.

In order to answer the above questions and overcome the limitations of scheduling studies using a real Grid testbed, we turned to simulation. While it is absolutely worthwhile to perform empirical evaluation of scheduling on the Grid, simulation-based investigations are valuable since they allow a controlled and repeatable evaluation for a range of scenarios: varying number of users, requirements, parameters, resources, and workload, etc. Towards this end, the next two chapters of this thesis discuss the GridSim toolkit we developed and the results of an extensive series of scheduling simulations. We also present comparative analysis of scheduling algorithms presented in this chapter along with a new cost-time optimisation algorithm that evolved during our simulation studies.

Software Availability

The Nimrod-G resource broker software with source code can be downloaded from the project website:

`http://www.csse.monash.edu.au/~davida/nimrod/`