SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2005; **35**:491–512
Published online 31 January 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.646

Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost–time optimization algorithm

Rajkumar Buyya[1,*,†], Manzur Murshed[2], David Abramson[3] and Srikumar Venugopal[1]

[1] *Grid Computing and Distributed Systems Laboratory,*
Department of Computer Science and Software Engineering, The University of Melbourne,
Parkville, Melbourne, VIC 3010, Australia
[2] *Gippsland School of Computing and Information Technology, Monash University, Gippsland Campus,*
Churchill, VIC 3842, Australia
[3] *School of Computer Science and Software Engineering, Monash University, Caulfield Campus,*
Melbourne, VIC 3145, Australia

SUMMARY

Computational Grids and peer-to-peer (P2P) networks enable the sharing, selection, and aggregation of geographically distributed resources for solving large-scale problems in science, engineering, and commerce. The management and composition of resources and services for scheduling applications, however, becomes a complex undertaking. We have proposed a computational economy framework for regulating the supply of and demand for resources and allocating them for applications based on the users' quality-of-service requirements. The framework requires economy-driven deadline- and budget-constrained (DBC) scheduling algorithms for allocating resources to application jobs in such a way that the users' requirements are met. In this paper, we propose a new scheduling algorithm, called the DBC cost–time optimization scheduling algorithm, that aims not only to optimize cost, but also time when possible. The performance of the cost–time optimization scheduling algorithm has been evaluated through extensive simulation and empirical studies for deploying parameter sweep applications on global Grids. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: Grid computing; scheduling; computational economy; Nimrod-G; Gridbus broker

*Correspondence to: Rajkumar Buyya, Grid Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Parkville, Melbourne, VIC 3010, Australia.
†E-mail: raj@cs.mu.oz.au

Received 19 December 2003
Revised 3 June 2004
Accepted 23 August 2004

1. INTRODUCTION

Computational Grids [1] and *peer-to-peer* (P2P) computing [2] networks are emerging as next-generation parallel and distributed computing platforms for solving large-scale computational and data-intensive problems in science, engineering, and commerce. They enable the *sharing*, *selection*, and *aggregation* of a wide variety of geographically distributed resources including supercomputers, storage systems, databases, data sources, and specialized devices owned by different organizations. However, resource management and application scheduling is a complex undertaking due to large-scale heterogeneity in resources, management policies, users, and application requirements in these environments [3].

A typical world-wide Grid computing environment is shown in Figure 1. In such a Grid marketplace and economy, the two key players are: *resource owners* (Grid service providers) and *end users* (Grid service consumers). The *resource owners* and *consumers/end-users* have different goals, objectives, strategies, and demand patterns. The resources are heterogeneous in terms of their architecture, power, configuration, and availability. They are owned and managed by different organizations with different access policies and cost models that vary with time, users, and priorities. Different applications have different computational models that vary with the nature of the problem.

In our earlier work [4–7], we investigated the use of economics as a metaphor for management of resources and scheduling applications in Grid computing environments. The computational economy framework provides a mechanism for regulating the supply of and demand for resources and allocating them to applications based on the users' quality-of-services (QoS) requirements [3]. It also offers an incentive to resource owners for sharing resources on the Grid, and offers end users a trade-off between the timeframe for result delivery and computational expenses.

A Grid scheduler, often called *resource broker*, acts as an interface between the user and distributed resources and hides the complexities of Grid computing [4,5]. It performs resource discovery, negotiates for access costs using trading services, maps jobs to resources (*scheduling*), stages the application and data for processing (*deployment*), starts job execution, and finally gathers the results. It is also responsible for monitoring and tracking the progress of application execution along with adapting to the changes in the runtime environment of the Grid, variation in resource share availability, and failures. Essentially, the Grid broker does application scheduling on distributed Grid resources on which it does not have full control—the local scheduler has its own policies and performs actual allocation of resource(s) to the user job(s).

The previous work in scheduling on distributed systems such as clusters and supercomputers has focused on extracting the maximum throughput from the entire system [8,9]. Grid scheduling, as shown by some of the related works in Section 2, concentrates on improving response times in an environment containing *autonomous* resources whose availability dynamically varies with time. The Grid scheduler has to interact with the local schedulers managing computational resources and adapt its behavior to changing resource loads. Thus the scheduling is conducted from the perspective of the application or the user rather than that of the system.

In our Grid economy framework, the resource brokers use economy-driven *deadline and budget-constrained* (DBC) scheduling algorithms for allocating resources to application jobs in such a way that the users' requirements are met. In our early work [7], we developed three scheduling algorithms for cost, time, and time-variant optimization strategies that support deadline and budget constraints.

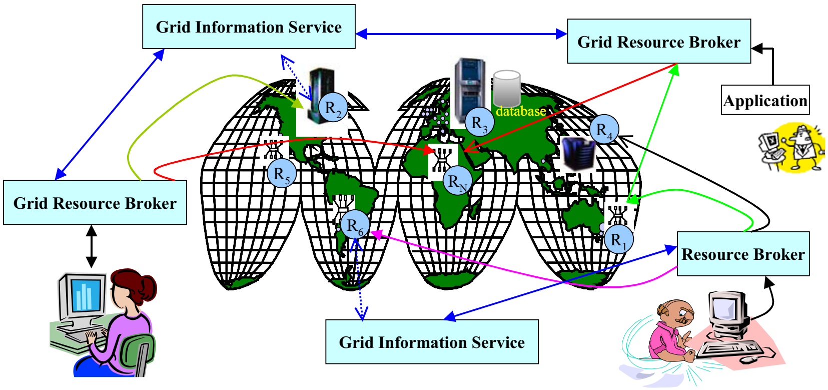


Figure 1. A generic view of the World-Wide Grid computing environment.

We implemented them within the Nimrod-G broker and explored their capability for scheduling task-farming or parameter-sweep applications such as drug design [10] on the World-Wide Grid (WWG) testbed resources [8]. To meet users' QoS requirements, the broker leases Grid resources and services dynamically at runtime depending on their capability, cost, and availability.

In this work, we propose a new scheduling algorithm, called *DBC cost–time optimization*, which extends the DBC cost-optimization algorithm to optimize for time, keeping the cost of computation at the minimum. Resources with the same cost are grouped together and a time-optimization scheduling strategy is applied while allocating jobs to a group. We demonstrate the ability of this new scheduling algorithm by implementing it within the economic Grid resource broker simulator built using the GridSim toolkit [11]. The performance of this new algorithm is evaluated by scheduling a synthetic task-farming application on simulated WWG testbed resources for different deadline and budget scenarios. We then compare and contrast the results of scheduling with the cost-optimization algorithm.

The rest of this paper is organized as follows. The related works in Grid resource management and scheduling are discussed in Section 2. Issues in scheduling applications on Grids along with a new DBC cost–time optimization scheduling algorithm proposed in this paper are discussed in Section 3. An economy-driven grid resource broker simulated using the GridSim toolkit and its internal components that simulate and manage the execution of task-farming applications are presented in Section 4. The simulation of heterogeneous resources with different capabilities and access costs, the creation of a synthetic application, and the evaluation, by simulation, of the proposed cost–time optimization scheduling algorithm versus the cost optimization algorithm is discussed in Section 5. The proposed algorithm is evaluated by empirical studies in Section 6. The final section presents the summary and conclusion.

2. RELATED WORK

A number of projects are investigating scheduling on distributed systems [3]. They include Grid resource management and scheduling systems such as Condor [12,13], Globus [14], Legion [15], AppLeS [16], NetSolve [17], and DISCWorld [18], which use system-centric scheduling strategies, and REXEC [19] and Spawn [20], which support computational economy-based resource management within cluster computing environments. The interest in computational economy is rapidly increasing within the Grid community [21] and projects such as G-Commerce [22] are exploring competitive resource pricing and allocation issues within the Grid environments.

The Application-Level Scheduling (AppLeS) project builds agents for each application responsible for offering a scheduling mechanism [16]. It uses a system-centric scheduling policy, which is targeted at minimizing the completion time—it does not take account of the economic cost of jobs processing while selecting resources. A recently developed AppLeS parameter-sweep template (APST) also uses system-centric scheduling strategies [23].

NetSolve is a client–agent–server system, which enables the user to solve complex scientific problems remotely [17]. The NetSolve agent does the scheduling by searching for those resources that offer the best performance in a network. The applications need to be built using one of the application programming interfaces (APIs) provided by NetSolve to perform remote procedure call (RPC)-like computations. NetSolve also provides an API for creating *task-farming* applications. The scheduling system maps jobs on resources that have appropriate libraries without taking the cost/price for processing jobs on them into consideration. Ninf [24] is an RPC implementation for grid computing similar to NetSolve.

Distributed Information Systems Control World (DISCWorld) is a service-oriented metacomputing environment, based on the client–server–server model [18]. Remote users can log in to this environment over the Internet and request access to data, and also invoke services or operations on the available data. DISCWorld aims for remote information access. The scheduling strategies used in the DISCWorld system are also system-centric in nature.

Another related tool that uses the concept of computational economy is REXEC, a remote execution environment [19] for co-operative distributed systems such as clusters with a centralized scheduling manager. At the command line, the maximum rate (credits per minute) that the user is willing to pay for CPU time can be specified. The REXEC client selects a node that fits the user's requirements and executes the application on it. The REXEC system provides an extended shell for remote execution of applications on clusters. Its scheduling strategies are targeted for centralized systems, and the allocation of resource share is proportional to the users' valuation of their jobs. In our Grid resource broker, scheduling strategies are targeted for geographically distributed systems—each resource has its own scheduler that performs actual allocation of resources to user jobs. That means cluster schedulers use cooperative computing economy since they aim for global optimization, whereas Grid schedulers use competitive computing economy since every entity aims to optimize its own objectives.

Spawn [20] is one of the earliest computational-economies-based resource-allocation systems. It uses a second-price auction model where tasks have to bid for computational time. Allocation of funding for tasks has to be performed by the application-level manager, which needs to be implemented by a programmer along with the necessary allocation strategies and priorities. That is, Spawn has mostly focused on developing an auction-based infrastructure for the computing marketplace and has left it to the programmers (application developers) to take care of allocation policies and algorithms. In fact, this complements the work described in this paper, as our focus is on developing a computational-economy-based application-scheduling system in addition to providing a parameter-sweep application-creation framework.

A number of works have explored the problem of deadline scheduling in real-time systems [25–27]. However, they mostly consider deadline parameters for individual jobs rather than groups. In addition, they have not addressed the concept of leasing third party services for meeting the deadline as they are restricted to centralized resource management schemes and single administrative domain resources.

3. DBC COST–TIME OPTIMIZATION SCHEDULING ALGORITHM

3.1. Application model and scheduling

The parameter-sweep application model has emerged as a 'killer application model' [4] for composing high-throughput computing (HTC) applications for processing on global Grids. This model is a combination of task and data parallel models and applications formulated using this model contain large number of independent jobs operating on 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 resembles 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. Fortunately, this high-throughput parametric computing model is simple, yet powerful enough to formulate distributed execution of many application areas such as: radiation equipment calibration analysis [4], searching for extra-terrestrial intelligence [28], protein folding [29], molecular modeling for drug design [10], human-genome sequence analysis [30], brain activity analysis, high-energy physics events analysis [31], *ad hoc* network simulation [32], crash simulation, tomography [33], financial modeling, and Mcell simulations [34]. Therefore, high-throughput parametric computing is considered as the killer application for the Grid.

Scheduling and orchestrating the execution of parameter-sweep applications on world-wide distributed computers appears simple, but complexity arises when users place QoS constraints such as execution completion time (deadline) and computation cost (budget) limitations along with optimization parameter preference. 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 organizations which have 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.

The integration of computational economy as part of a scheduling system greatly influences the way computational resources are selected to meet the users' requirements. A user 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.

3.2. A new scheduling algorithm

We have developed a number of algorithms for DBC scheduling of task-farming parameter-sweep applications on the Grid. They are: cost-optimization, time-optimization, and conservative time-optimization scheduling algorithms. The performance of these algorithms has been evaluated by implementing them within the Nimrod-G resource broker [11] for scheduling real-world applications and economic-broker simulators [7] through synthetic workloads.

In this paper, we propose a new DBC Grid scheduling algorithm, called the cost–time optimization scheduling algorithm, which builds on the cost-optimization and time-optimization scheduling algorithms. This is accomplished by applying the time-optimization algorithm to schedule task-farming application jobs on distributed resources having the same processing cost. A detailed algorithm for mapping jobs to resources based on this new strategy is listed in Figure 2. An application containing N jobs along with the deadline and budget constraints are passed as input parameters to the algorithm for processing on M distributed resources/machines. Essentially, the user passes these details to the Grid broker that leases resources dynamically at runtime depending on their availability, capability, cost, and users' QoS requirements. In the next subsection, we discuss methods for evaluation of this new algorithm, and later sections present detailed evaluation.

3.3. Evaluation methods and complexity analysis

A variety of techniques and technologies exist for carrying out performance evaluation of resource-management and scheduling algorithms. Some evaluation techniques include: analytical, simulation, emulation and empirical. Some of the notable Grid tools are SimGrid [35] and GridSim [11] for *simulation*; MicroGrid [36] for *emulation*; and Nimrod-G [5] that supports creation of pluggable schedulers for *empirical* evaluation. In addition, empirical evaluation needs system-level Grid middleware such as Globus for deploying jobs securely on Grid testbed resources.

Through analysis of the algorithm listed in Figure 2, it can be seen that the loop in step 4(e) determines the time complexity. This loop costs 1 when there is 1 resource group (i.e. when all M machines are of the same cost) and M when there are M resource groups (i.e. all machines have different costs). Within this loop, the loop in step 4(i) costs N. Within the second loop, the step leading to job matching costs is M for 1 resource group and 1 for M resource groups.

Bringing it all together, the overall complexity can be calculated as:

- $\mathrm{O}(1 * (N * (M)) = \mathrm{O}(NM)$ when all resources have the same cost; and
- $\mathrm{O}(M * (N * (1))) = O(NM)$ when all resources have different costs.

The cost-optimization algorithm also has a time complexity of $\mathrm{O}(NM)$. However, as we will show in later sections, the cost–time-optimization algorithm processes the jobs in less time than the cost-optimization algorithm while using the same amount of budget.

Further on in this paper, we use simulation and empirical methods to evaluate the performance of an economic-based resource allocation strategy. Simulation allows the creation of large-scale virtual Grid environments, and usage and availability scenarios that can be repeated and controlled. We have selected GridSim for simulation as it supports modeling and simulation of a variety of resources—shared and distributed memory machines, managed as time- or space-shared resources. To evaluate the feasibility of the proposed scheduling strategy, we have implemented a plug-in scheduler for the Nimrod-G Grid resource broker. However, it should be noted that in a real Grid testbed, it is impossible to conduct *repeatable* and *comparable* evaluations as the availability of resources varies with time and there is no centralized control to create a stable environment. Sections 5 and 6 present the results of evaluation through simulation and empirical methods respectively.

Algorithm: *DSC_Scheduling_with_Cost_Time_Optimization* (*Application*: *N jobs*, *Resources*: *M*, *Deadline*: *D*, *Budget*: *B*)

(1) RESOURCE DISCOVERY: Identify characteristics, configuration, capability, and suitability of resources using the Grid information services (GIS).

(2) RESOURCE TRADING: Identify *the cost* of all resources and *the capability* to be delivered per cost-unit. The *resource cost* can be expressed in units such as processing cost per Million Instructions (MI) cost per job, CPU cost per time unit etc., and the scheduler needs to choose suitable units for comparison.

(3) If the user supplies *d* and *b* factors, then determine the absolute deadline and budget based on the capability of resources and their cost, and the application processing requirement (e.g. total MI required).

(4) SCHEDULING: Repeat while there exist *unprocessed jobs* and the current time and processing expenses are within the deadline and budget limits. [This step is triggered for each scheduling event or whenever a job completes. The event period is a function of deadline, job processing time, rescheduling overhead, resource share variation, etc.]
[SCHEDULE ADVISOR with Policy]

- (a) For each resource, predict and establish the *job consumption rate* or *the available resource share* through the measure and extrapolation strategy taking into account the time taken to process previous jobs.
- (b) SORT the resources by increasing order of *cost*. If two or more resources have the *same cost*, order them such that powerful ones (e.g. higher job consumption rate or resource share availability, but, the first time, based on the total theoretical capability, say the total Million Instructions per second (MIPS) are preferred first.
- (c) *Create resource groups* containing resources with the same cost.
- (d) If any of the resource has jobs assigned to it in the previous scheduling event, but not dispatched to the resource for execution and there is variation in resource availability, then move appropriate number of jobs to the Unassigned-Jobs-List. This helps in updating the whole schedule based on the latest resource availability information.
- (e) Repeat the following steps for each resource group as long as there exist unassigned jobs:
    - (i) *Repeat the following steps for each job in the Unassigned-Jobs-List depending on the processing cost and the budget availability*: [It uses the *time optimization strategy*.]
        - Select a job from the Unassigned-Jobs-List.
        - For each resource, calculate/predict the job completion time taking into account previously assigned jobs and the job completion rate and resource share availability.
        - Identify the resource with the least completion time and assign the job to it provided it is able to complete the job within the deadline. Remove the assigned job from the Unassigned-Jobs-List.

(5) [DISPATCHER with Policy]
Repeat the following steps for each resource if it has jobs to be dispatched:

- Identify the number of jobs that can be submitted without overloading the resource. Our default policy is to dispatch jobs as long as the number of user jobs deployed (active or in queue) is less then the number of PEs (Processing Elements, i.e. CPUs) in the resource.

Figure 2. DBC scheduling with cost–time optimization.

4. IMPLEMENTATION OF ECONOMIC GRID BROKER SIMULATOR

The GridSim toolkit is used to simulate a Grid environment and a Nimrod-G-like deadline and budget constrained scheduling system called economic Grid resource broker. The simulated Grid environment contains multiple resources and user entities with different requirements. The user and broker entities extend the GridSim class. All the users create experiments containing application specification requirements (a set of Gridlets that represent application jobs) and QoS requirements (deadline and budget constraints with optimization strategy). When the simulation starts, the user entity creates an instance of its own broker entity and passes a request for processing application jobs. We briefly discuss features of the GridSim toolkit and its usage in the implementation of the economic broker simulator that supports performance evaluation of scheduling algorithms.

4.1. GridSim: a Grid modeling and simulation toolkit

The GridSim toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers [11]. It has facilities for the modeling and simulation of resources and network connectivity with different capabilities, configurations, and domains. It supports primitives for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution. These features can be used to simulate resource brokers or Grid schedulers for evaluating performance of scheduling algorithms or heuristics. We have used the GridSim toolkit to create a resource broker that simulates Nimrod-G for design and evaluation of deadline and budget constrained scheduling algorithms with cost and time optimizations.

The GridSim toolkit resource modeling facilities are used to simulate the World-Wide Grid resources managed as time- or space-shared scheduling policies. The broker and user entities extend the GridSim class to inherit ability for communication with other entities. In GridSim, application tasks/jobs are modeled as *Gridlet* objects that contain all the information related to the job and its execution management details such as job length in MI, disk input/output operations, input and output file sizes, and the job originator. The broker uses GridSim's job management protocols and services to map a Gridlet to a resource and manage it throughout its lifecycle. The broker also maintains full details of application scheduling trace data both at coarse and fine levels, which can be used in performance analysis.

4.2. Economic Grid broker simulator architecture

The broker entity architecture and its interaction with other entities is shown in Figure 3. The key components of the broker are: experiment interface, resource discovery and trading, scheduling flow manager backed with scheduling heuristics and algorithms, Gridlets dispatcher, and Gridlets receptor. A detailed discussion of the broker implementation using the GridSim toolkit can be found in Buyya and Murshed [11]. However, to enable the understanding of the broker framework in which the new scheduling algorithm is implemented, we briefly present its operational model.

(1) The user entity creates an experiment that contains the application description (a list of Gridlets to be processed) and sends user requirements to the broker via the experiment interface.

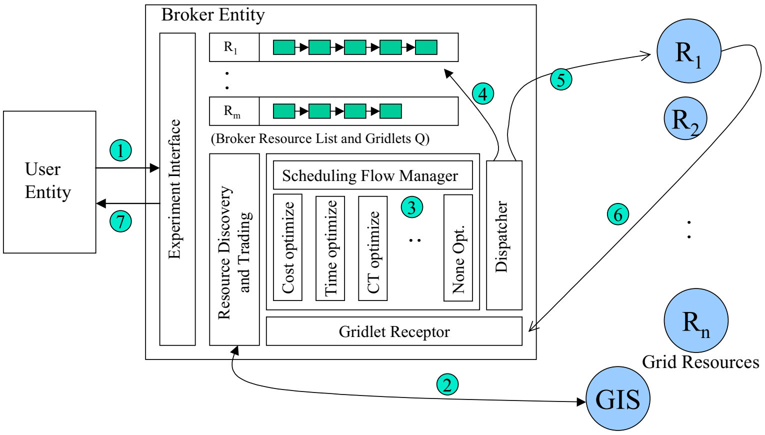


Figure 3. Economic Grid resource broker architecture and its interaction with other entities.

(2) The broker resource discovery and trading module interacts with the GridSim GIS entity to identify the contact information of the resources, and then interacts with resources to establish their configuration and access cost. It creates a broker resource list that acts as a placeholder for maintaining resource properties, a list of Gridlets committed for execution on the resource, and the resource performance data as predicted through the measurement and extrapolation methodology.

(3) The scheduling flow manager selects an appropriate scheduling algorithm for mapping Gridlets to resources depending on the user's requirements (deadline and budget limits, and optimization strategy—cost, cost–time, time, and conservative-time). Gridlets that are mapped to a specific resource are added to the Gridlets list in the broker resource. In this case, the broker selects the algorithm for DBC cost–time-optimization scheduling.

(4) For each of the resources, the dispatcher selects the number of Gridlets that can be staged for execution according to the usage policy to avoid overloading resources with single user jobs.

(5) The dispatcher then submits Gridlets to resources using the GridSim's asynchronous service.

(6) When the Gridlet processing completes, the resource returns it to the broker's Gridlet receptor module, which then measures and updates the runtime parameter, *resource or MI share available to the user*. It aids in predicting the job consumption rate for making scheduling decisions.

(7) Steps 3–6 continue until all the Gridlets are processed or the broker exceeds the deadline or budget limits. At the end, the broker returns updated experiment data along with processed Gridlets back to the user entity.

5. SIMULATION-BASED PERFORMANCE EVALUATION

To simulate and evaluate application scheduling in a GridSim environment using the economic Grid broker requires the modeling and creation of GridSim resources and applications that model jobs as Gridlets. In this section, we present resource and application modeling along with the results of scheduling experiments with QoS-driven application processing.

5.1. Resource modeling

We modeled and simulated a number of time- and space-shared resources with different characteristics, configurations, and capability as those in the World-Wide Grid testbed. We have selected the latest CPU models AlphaServer ES40, Sun Netra 20, Intel VC820 (800EB MHz, Pentium III), and SGI Origin 3200 1X 500 MHz R14k, released by their manufacturers Compaq, Sun, Intel, and SGI respectively. The processing capability of these PEs in simulation time-units is modeled after the base value of the SPEC CPU (INT) 2000 benchmark ratings [37]. To enable the users to model and express their application processing requirements in terms of MI or MIPS on the standard machine, we *assume the MIPS rating of PEs is same as the SPEC rating*.

Table I shows the characteristics of the resources simulated and their PE access cost per time unit in Grid dollars (G$). The PE capability of resources is derived from their actual SPEC rating and access cost in G$ is artificially assigned. The simulated resources resemble the World-Wide Grid testbed resources used in the Nimrod-G scheduling experiments [38]. The access cost of a PE in G$/time unit does not necessarily reflect the cost of processing when PEs have different capabilities. The brokers need to translate the access cost into the G$/MI for each resource. Such translation helps in identifying the relative cost of resources for processing Gridlets on them. It can be noted some of the resources in Table I have the same MIPS/G$, for example, R4 and R8.

5.2. Application modeling

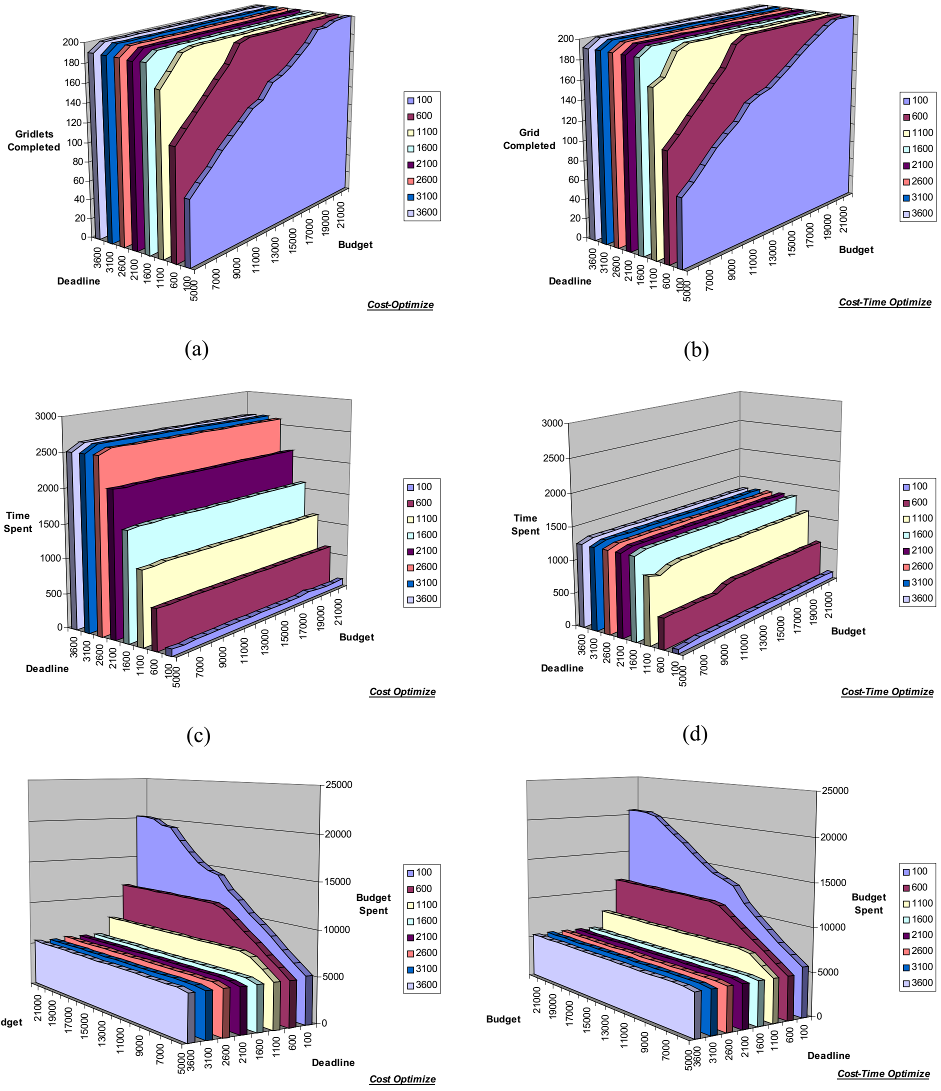
We have modeled a task-farming application that consists of 200 jobs. In GridSim, these jobs are packaged as Gridlets whose contents include the job length in MI, the size of job input, and output data in bytes, along with various other execution-related parameters when they move between the broker and resources. The job length is expressed in terms of the time it takes to run on a standard resource PE with SPEC/MIPS rating of 100. Gridlets' processing time is expressed in such a way that they are expected to take at least 100 time units with a random variation of 0–10 per cent on the positive side of the standard resource. That means Gridlets' job length (processing requirements) can be at least 10 000 MI with a random variation of 0–10 per cent on the positive side. This 0–10 per cent random variation in Gridlets' job length is introduced to model heterogeneous tasks similar to those present in the real-world parameter-sweep applications.

5.3. Scheduling experiments with cost- and cost–time-optimization strategies

We performed both cost- and cost–time optimization scheduling experiments with different DBC values for a single user. The deadline is varied in simulation time from 100 to 3600 in steps of 500. The budget is varied from G$ 5000 to 22 000 in steps of 1000. The number of Gridlets processed, deadline utilized, and budget spent for the DBC cost-optimization scheduling strategy is shown in Figures 4(a), 4(c), and 4(e), and for the cost–time-optimization scheduling strategy in Figures 4(b), 4(d), and 4(f). In both cases, when the deadline is low (e.g. 100 time units), the number of Gridlets processed increases as the budget value increases. When a higher budget is available, the broker leases

Table I. World-Wide Grid testbed resources simulated using GridSim.

Resource name in simulation	Reference resource characteristics: vendor, type, node OS, no. of PEs	Equivalent resource in World-Wide Grid (hostname, location)	A PE SPEC/MIPS rating	Resource manager type	Price (G$/PE time unit)	SPEC/MIPS per G$
R0	Compaq, AlphaServer, CPU, OSF1, 4	grendel.vpac.org, VPAC, Melb, Australia	515	Time-shared	8	64.37
R1	Sun, Ultra, Solaris, 4	hpc420.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	4	94.25
R2	Sun, Ultra, Solaris, 4	hpc420-1.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	3	125.66
R3	Sun, Ultra, Solaris, 2	hpc420-2.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	3	125.66
R4	Intel, Pentium/VC820, Linux, 2	barbera.cnuce.cnr.it, CNR, Pisa, Italy	380	Time-shared	1	380.0
R5	SGI, Origin 3200, IRIX, 6	onyx1.zib.de, ZIB, Berlin, Germany	410	Time-shared	5	82.0
R6	SGI, Origin 3200, IRIX, 16	Onyx3.zib.de, ZIB, Berlin, Germany	410	Time-shared	5	82.0
R7	SGI, Origin 3200, IRIX, 16	mat.ruk.cuni.cz, Charles U., Prague, Czech Republic	410	Space-shared	4	102.5
R8	Intel, Pentium/VC820, Linux, 2	marge.csm.port.ac.uk, Portsmouth, U.K.	380	Time-shared	1	380.0
R9	SGI, Origin 3200, IRIX, 4 (accessible)	green.cfs.ac.uk, Manchester, U.K.	410	Time-shared	6	68.33
R10	Sun, Ultra, Solaris, 8,	pitcairn.mcs.anl.gov, ANL, Chicago, U.S.A.	377	Time-shared	3	125.66



(a)

(b)

(c)

(d)

(e)

(f)

Figure 4. The number of Gridlets processed, time, and budget spent for different deadline and time limits when scheduled using the cost- and cost–time-optimization algorithms: (a) no. of Gridlets processed; (b) no. of Gridlets processed; (c) time spent for processing Gridlets; (d) time spent for processing Gridlets; (e) budget spent for processing Gridlets; (f) budget spent for processing Gridlets.

expensive resources to process more jobs within the deadline. Alternatively, when scheduling with a low budget, the number of Gridlets processed increases as the deadline is relaxed.

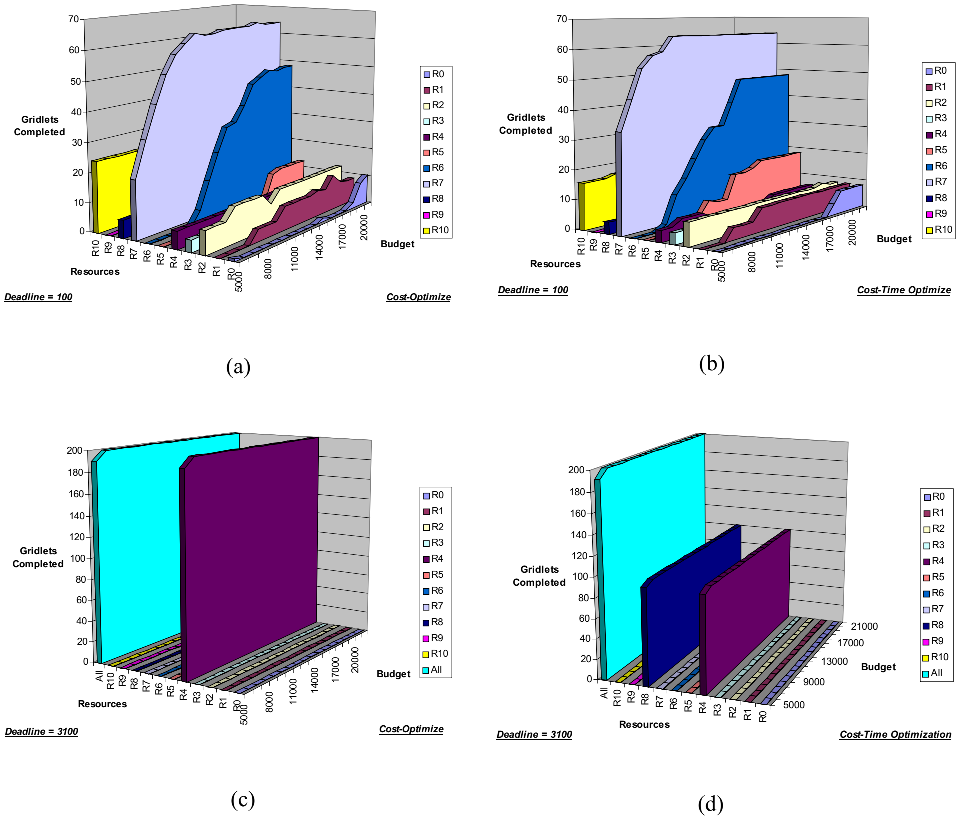
The impact of budget for different values of deadline is shown in Figures 4(e) and 4(f) for cost and cost–time strategies. For a larger deadline value (see the time utilization for a deadline of 3600), the increase in budget value does not have much impact on resource selection. When the deadline is too tight (e.g. 100), it is likely that the complete budget is spent processing Gridlets within the deadline.

It can be observed that the number of Gridlets processed and the budget-spending pattern is similar for both scheduling strategies. However, the time spent for the completion of all the jobs is significantly different (see Figures 4(c) and 4(d)), as the deadline becomes relaxed. For deadline values from 100 to 1100, the completion time for both cases is similar, but as the deadline increases (e.g. from 1600 to 3600), the experiment completion time for the cost–time-scheduling optimization strategy is much less than that for the cost-optimization scheduling strategy. This is because when there are many resources with the same MIPS/G$, the cost–time-optimization scheduling strategy allocates jobs to them using the time-optimization strategy for the entire deadline duration since there is no need to spent extra budget for doing so. This does not happen in case of the cost-optimization strategy—it allocates as many jobs as the first cheapest resource can complete by the deadline, and then allocates the remaining jobs to the next cheapest resources.

A trace of resource selection and allocation using cost- and cost–time-optimization scheduling strategies, shown in Figure 5, indicates their impact on the application processing completion time. When the deadline is tight (e.g. 100), there is high demand for all the resources in a short time; the impact of cost- and cost–time-scheduling strategies on the completion time is similar as all the resources are used up so long as budget is available to process all jobs within the deadline (see Figures 5(a) and 5(b)). However, when the deadline is relaxed (e.g. 3100), it is likely that all jobs can be completed using the first few cheapest resources. In this experiment there were resources with the same cost and capability (e.g. R4 and R8). The cost-optimization strategy selected resource R4 to process all the jobs (see Figure 5(c)), whereas the cost–time-optimization strategy selected both R4 and R8 (see Figure 5(d)) since both resources cost the same price, and completed the experiment earlier than the cost-optimization scheduling (see Figures 4(c) and 4(d)). This situation can be observed clearly in scheduling experiments with a large budget for different deadline values (see Figure 6). Note that the left-most solid curve marked ‘All’ in the Resources axis in Figure 6 represents the aggregation of all resources.

As the deadline increases, the cost-optimization algorithm predominantly schedules jobs on the resource R4 (see Figure 6(a)), whereas the cost–time-optimization algorithm scheduled jobs on resources R4 and R8 (see Figure 6(b)), the first two cheapest resources with the same cost. Therefore, the application scheduling using the cost–time-optimization algorithm is able to finish earlier compared to the one scheduled using the cost-optimization algorithm (see Figure 7), and both strategies spend the same amount of budget processing their jobs (see Figure 8). The completion time for the cost-optimization scheduling continued to increase with increase of the deadline as the broker allocated more jobs to resource R4 and less to resource R8. However, the completion time for deadline values 3100 and 3600 was the same as 2600 since the broker allocated jobs to only resource R4. This was not the case with the cost–time-optimization scheduling since jobs were allocated proportionally to both resources R4 and R8 and thus minimizing the completion time without spending any extra budget.

A microscopic evaluation of mapping of jobs to different resources under the *single user* and the *multiple competing users* scenarios for the entire execution period can be found in [3].



(a)

(b)

(c)

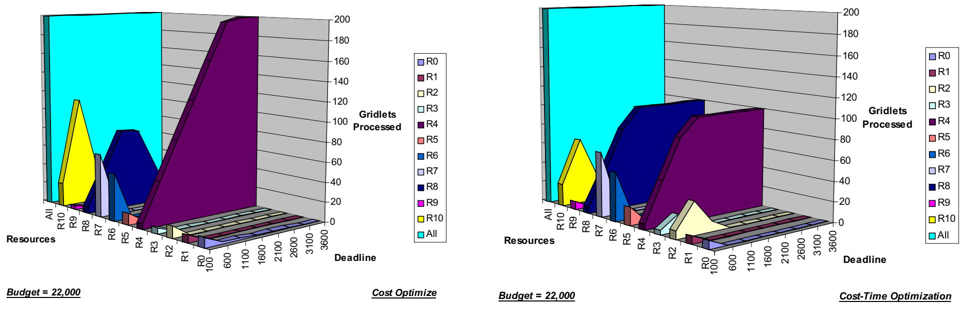
(d)

Figure 5. The number of Gridlets processed and resources selected for different budget values with a long deadline value when scheduled using the cost- and cost–time-optimization algorithms: (a) cost optimization with a short deadline; (b) cost–time optimization and a short deadline; (c) cost optimization with a long deadline; (d) cost–time optimization with a long deadline.

6. EMPIRICAL EVALUATION—SCHEDULING ON THE WORLD-WIDE GRID TESTBED

The aim of this empirical study is to demonstrate that it is feasible to implement and deploy the cost–time-optimization algorithm for scheduling applications on real Grid resources. The list of resources[‡] used in application scheduling experiments drawn from the SC2002 Global Grid Testbed

[‡] The resources used in this empirical study are different from those simulated (reference) resources as they were either inaccessible due to middleware incompatibility problems (e.g. Globus Monitoring and Discovery System software is not backwards compatible), or, in some cases, we lost access due to change of access policies.



(a) (b)

Figure 6. The number of Gridlets processed and resources selected for different deadline values with a high budget when scheduled using the cost- and cost–time-optimization algorithms: (a) resource selection in cost optimization scheduling when the budget is high; (b) resource selection in cost–time optimization scheduling when the budget is high.

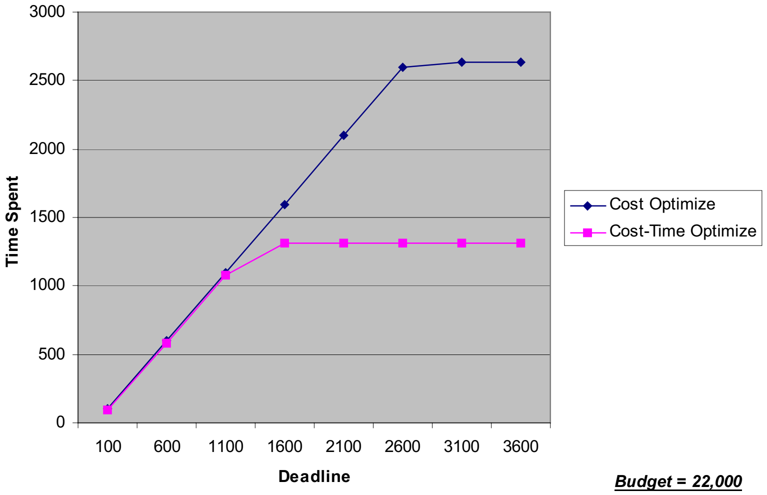
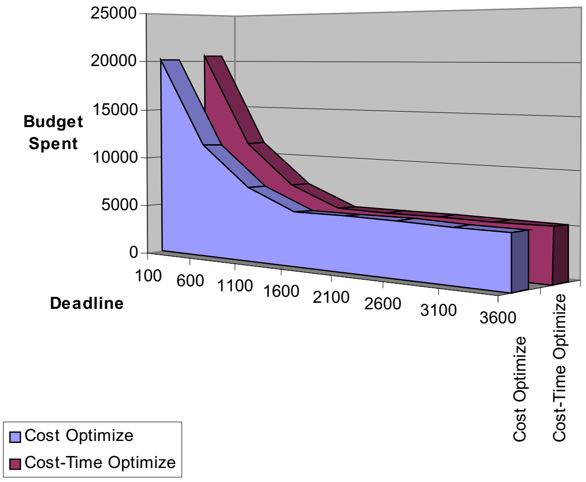


Figure 7. The time spent for processing application jobs for different deadline constraints with a large budget when scheduled using the cost- and cost–time-optimization algorithms.



Budget = 22,000

Figure 8. The budget spent for processing application jobs for different deadline constraints with a large budget when scheduled using the cost- and cost–time-optimization algorithms.

Collaboration [39] and those of the WWG [3] is given in Table II. As these distributed resources are shared among various users and no single user has control over allocation of resources, it is *impossible* to carry out *repeatable* and *comparable* evaluations as the availability of resources varies with time and there is no centralized control to create a stable environment. This situation has been noted in many early works [11,16,35] and also in empirical results presented in the rest of the section.

The cost–time-optimization strategy has been *recently* implemented through the Gridbus [40] scheduler, which was developed as a plug-in scheduler to the Nimrod-G broker using its APIs. The API implements a string-based protocol which allows a program to steer a computation through Nimrod, but in its own way rather than use the Nimrod scheduling algorithms. This way of testing experimental optimization strategies does not require Nimrod-G itself to be changed. Here, Nimrod-G performs resource discovery, selection using Gridbus schedule, and dispatching of jobs to a remote resource, starting and managing the execution of jobs and gathering the results back at the home node.

The costs shown in the Table II for each node were assigned artificially for this experiment only. However, the scheduler would find out the cost of each node from the Grid Market Directory [41] based on the GRACE trading protocols (commodity market models). Secure and remote access to all these resources is enabled through Globus middleware.

An experiment consisted of scheduling a parameter-sweep application for execution on the various nodes in our Grid testbed using Nimrod-G. The application used here is *calc*, a program that calculates mathematical functions based on the values of two input parameters. The first parameter, *length*, is an input to a mathematical function and the second parameter, *time_base_value*, indicates the expected calculation complexity in minutes plus 0–60 minutes. A plan file modeling this application

Table II. List of Grid resources used in the experimentation.

Organization	Node details (architecture, no. of nodes, hostname)	Cost (G$ per CPU sec)

N*Grid Project Korea	`Linux Cluster, 24 nodes`
	`node1001.gridcenter.or.kr`
		3

Vrije Universiteit, Netherlands	`Linux Cluster, 144 nodes`
	`(32 available),fs0.das2.cs.vu.nl`
		2

N*Grid Project Korea	`Linux Cluster, 16 nodes,`
	`node2001.gridcenter.or.kr`
		1

IIT, NRC, Canada	`IBM SP, 4 nodes,`
	`hpc76.ai.iit.nrc.ca`
		1

Department of Physics, University of Melbourne, Australia	`Linux Cluster, 10 nodes,`
	`lem.ph.unimelb.edu.au`
		1

Cambridge University, U.K.	`Linux Cluster, 20 nodes,`
	`herschel.amtp.cam.ac.uk`
		1

MARCC, University of Melbourne, Australia	`Linux Cluster, 8 nodes,`
	`gnet01.hpc.unimelb.edu.au`
		1

```
#Parameter definition
parameter length integer range from 1 to 200 step 1;
parameter time_base_alue integer default 10;
#Task definition
task definition
task main
        #Copy necessary executables depending on node type
        copy clac.SOS node:calc
        #Execute program with parameter values on remote node
        node:execute ./calc $length $time_base_value
        #Copy results file to use home node with jobname as extension
        copy node:output ./output.$jobname
endtask
```

Figure 9. Plan file for executing calc application using Nimrod-G parameter-specification language.

as a parameter sweep application using the Nimrod-G parameter-specification language is shown in Figure 9. The first part defines parameters and the second part defines the task that is to be performed for each job. As the parameter *length* varies from values 1 to 200 in steps of 1, this plan file would create 200 jobs with input values from 1 to 200. To execute each job on a Grid resource, Nimrod-G copies the program executable to a Grid node, executes the program and finally copies the results back to the user home node and stores results in the output file with jobs number as file extension.

The experiments were carried out on 8 December 2002 between 10 a.m. and 1 p.m. Australian Eastern Daylight Time. The cost- and cost–time-optimization strategies were tried out and compared.

Table III. Summary of experiment statistics.

Scheduling strategy	Start time	Completion time	Execution time (min)	Budget consumed (G$)
Cost	10:00 a.m.	11:27 a.m.	87	188
Cost–time	11:40 a.m.	12:08 p.m.	28	277

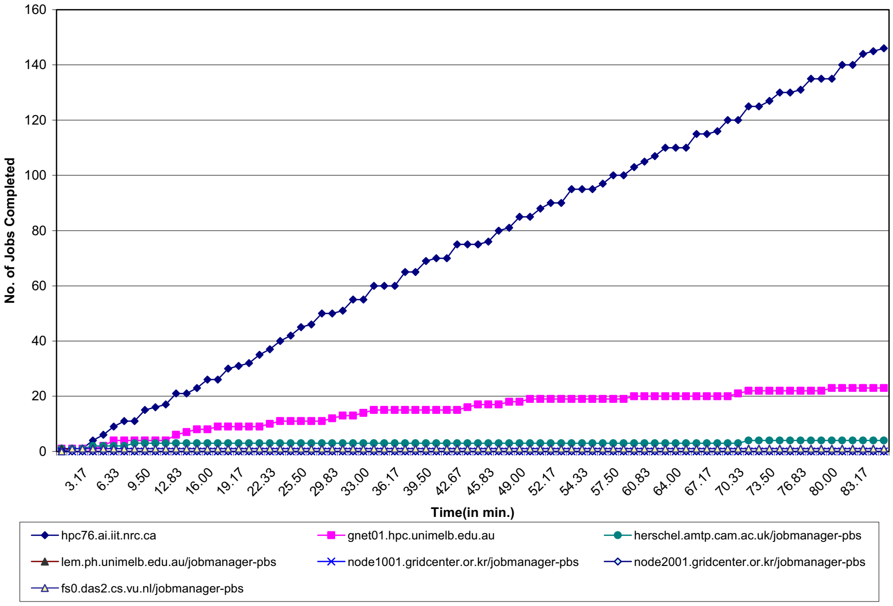


Figure 10. Cumulative graph of no. of jobs completed versus time for cost-optimization strategy.

All experiments were started with: deadline = 2 hours, budget = 600 G$. The summary of the results of these experiments is given in Table III.

The graph in Figure 10 shows cumulatively the number of jobs done against time taken. Here, we can see that the scheduler has allocated most of the jobs to the machine with the least cost per CPU cycle (i.e. hpc76.iit.nrc.ca). It does allocate some jobs to another machine (gnet01.hpc.unimelb.edu.au) so that the deadline can be achieved. Based on the performance of the first node, the scheduler realizes that it can finish the jobs within the deadline. Hence most of the jobs are allocated to the Canadian IBM-SP. Thus we see that given a fairly relaxed deadline, as in this case, the scheduler tries to execute the jobs in the least expensive way using the cost-optimization strategy.

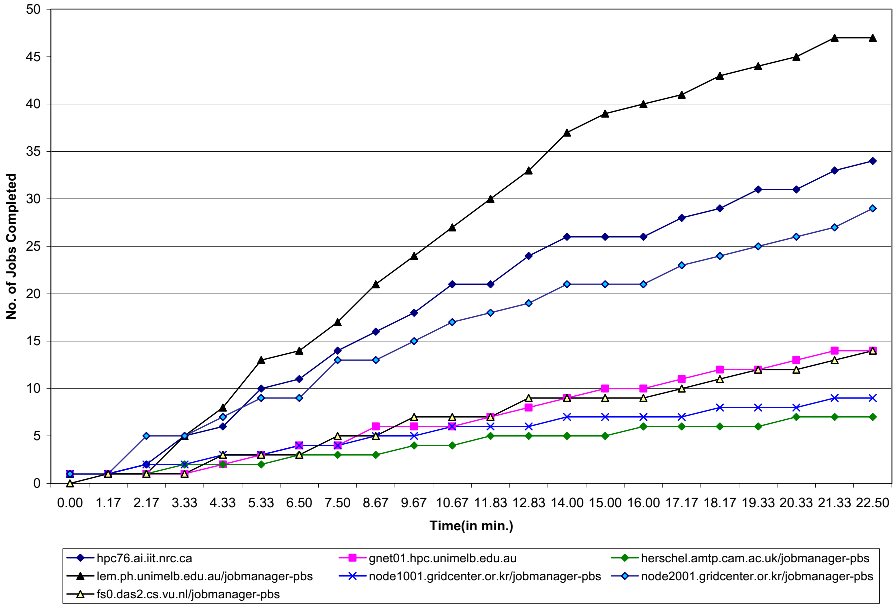


Figure 11. Cumulative graph of no. of jobs completed versus time for cost–cost-optimization.

The cumulative graph for cost–time optimization in Figure 11 shows a different case. Here, most of the jobs are executed by the three least expensive machines (lem.ph.unimelb.edu.au, node2001.gridcenter.or.kr, hpc76.ai.nrc.ca). It can be seen that, as predicted by the simulation, the scheduler performs time optimization among the machines with the least costs, and hence the University of Melbourne Physics cluster is allocated the maximum number of jobs as it is the fastest among the cheapest. However, the scheduler does allocate some jobs in the beginning to other, more expensive machines as initially two of the cheaper machines (lem.ph.unimelb.edu.au and hpc76.ai.iit.nrc.ca) are slow to pick up jobs.

In terms of the budget spent during execution, there is a deviation between the results of simulation and the actual experimental results. Simulation predicts that with a relaxed deadline, the budget spent should nearly be the same for both the cost–time- and cost-optimization algorithms. However, in these experiments, the cost–time-optimization method is found to be more expensive than the cost-optimization method (see Table III). This is due to a huge variation in the availability of some of the Grid resources during the cost–time-optimization scheduling. For example, the Canadian IBM SP machine was able to process many jobs during the cost-optimization scheduling experiment, but the available processing capability had reduced during the cost–time-optimization. This impacted on the amount of budget that was being spent. If the least expensive machines had performed well in the beginning, the cost would have remained the same.

7. SUMMARY AND CONCLUSION

Computational Grids enable the sharing, discovery, selection, and aggregation of geographically distributed heterogeneous resources for solving large-scale applications. We proposed computational economy as a metaphor for managing the complexity that is present in the management of distributed resources and their allocation. It allows allocation of resources depending on the users' QoS requirements such as the deadline, budget, and optimization strategy. This paper proposed a new deadline- and budget-constrained scheduling algorithm called *cost–time optimization*. We developed a scheduling simulator using the GridSim toolkit and evaluated the new scheduling algorithm by comparing its performance and quality of service delivery with the cost-optimization algorithm.

When there are multiple resources with the same cost and capability, the cost–time-optimization algorithm schedules jobs on them using the time-optimization strategy for the deadline period. From the results of scheduling experiments for many scenarios with a different combination of deadline and budget constraints, we observe that applications scheduled using the *cost–time*-optimization algorithm are able to complete earlier than those scheduled using the cost-optimization algorithm, without incurring any extra expenses. This establishes the superiority of the new deadline- and budget-constrained cost–time-optimization algorithm in scheduling jobs on global Grids. The cost–time-optimization strategy has also been implemented in the Gridbus scheduler to demonstrate that it is feasible to implement and deploy the cost–time-optimization algorithm for scheduling applications on global Grids.

SOFTWARE AVAILABILITY

The GridSim toolkit and the economic Grid broker simulator with source code can be downloaded from: http://www.gridbus.org/gridsim/.

The Nimrod-G broker can be downloaded from: http://www.csse.monash.edu.au/~davida/nimrod/.

The Gridbus broker can be downloaded from: http://www.gridbus.org/broker/.

ACKNOWLEDGEMENTS

We thank Rob Gray (DSTC) for his comments on improving the paper and Marcelo Pasin (Federal University of Santa Maria, Brazil) for his help in modeling resources with SPEC benchmark ratings. We would like to thank participants in the SC2002 Global Grid Testbed Collaboration for providing us with access to the resources that have made large-scale experiments possible.

We are grateful to the reviewers for their meticulous and constructive comments that helped us in improving the quality of this paper substantially.

REFERENCES

1. Foster I, Kesselman C (eds.). *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers: San Francisco, CA, 1999.
2. Oram A. (ed.). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly Press, 2001.
3. Buyya R. Economic-based distributed resource management and scheduling for Grid computing. *PhD Thesis*, Monash University, Melbourne, 2002.

4. Abramson D, Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: Killer application for the global Grid? *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2000)*. Cancun, Mexico, 1–5 May 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.
5. Buyya R, Abramson D, Giddy J. Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid. *Proceedings of the 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000)*, Beijing, China, May 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.
6. Buyya R, Abramson D, Giddy J. An economy driven resource management architecture for global computational power Grids. *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, June 2000. CSREA Press, 2000.
7. Buyya R, Giddy J, Abramson D. An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications. *Proceedings of the 2nd International Workshop on Active Middleware Services (AMS 2000)*, Pittsburgh, PA, August 2000. Kluwer, 2000.
8. Casavant TL, Kuhl JG. A taxonomy of scheduling in general purpose distributed computing. *IEEE Transactions on Software Engineering* 1988; **14**(2).
9. Feitelson DG, Rudolph L (eds.). *Proceedings of the 5th IPPS/SPDP'99 Workshop Job Scheduling Strategies for Parallel Processing (JSSPP 1999)*, San Juan, Puerto Rico, April 1999 (*Lecture Notes in Computer Science*, vol. 1659). Springer: Heidelberg, 1999.
10. Buyya R, Branson K, Giddy J, Abramson D. The virtual laboratory: Enabling molecular modelling for drug design on the World Wide Grid. *Technical Report CSSE-103*, Monash University, 2001.
11. Buyya R, Murshed M. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience* 2002; **14**(13–15).
12. Litzkow M, Livny M, Mutka M. Condor—a hunter of idle workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988. Editrice Compositori/Press: Bologna, Italy, 1988.
13. Frey J, Tannenbaum T, Foster I, Livny M, Tuecke S. Condor-G: A computation management agent for multi-institutional Grids. *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, San Francisco, CA, 7–9 August 2001.
14. Czajkowski K, Foster I, Karonis N, Kesselman C, Martin S, Smith W, Tuecke S. A resource management architecture for metacomputing systems. *Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1998.
15. Weissman J, Grimshaw A. A federated model for scheduling in wide-area systems. *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing (HPDC)*. Sage Publications: Thousand Oaks, CA, 1996.
16. Berman F, Wolski R. The AppLeS project: A status report. *Proceedings of the 8th NEC Research Symposium*, Germany, May 1997. Elsevier Press: Amsterdam, The Netherlands, 1997.
17. Casanova H, Kim M, Plank J, Dongarra J. Adaptive scheduling for task farming with Grid middleware. *The International Journal of High Performance Computing* 1999; **13**(3).
18. Hawick K *et al.* DISCWorld: An environment for service-based metacomputing. *Future Generation Computing Systems* 1999; **15**.
19. Chun B, Culler D. User-centric performance analysis of market-based cluster batch schedulers. *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, Berlin, Germany, May 2002. CERN: Geneva, Switzerland, 2002.
20. Waldspurger C, Hogg T, Huberman B, Kephart J, Stornetta W. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, February 1992.
21. Hacker T, Thigpen W. Accounting models research group. *Global Grid Forum*. http://www.gridforum.org/5_ARCH/ACCT.htm.
22. Wolski R, Plank J, Brevik J, Bryan T. Analyzing market-based resource allocation strategies for the computational Grid. *International Journal of High-performance Computing Applications* 2001; **15**(3).
23. Casanova H, Obertelli G, Berman F, Wolski R. The AppLeS parameter sweep template: User-level middleware for the Grid. *Proceedings of the IEEE SC 2000: International Conference Networking and Computing*, Dallas, TX, November 2000. TurboWorx: Shelton, CT, 2000.
24. Sato M, Nakada H, Sekiguchi S, Matsuoka S, Nagashima U, Takagi H. Ninf: A network based information library for global world-wide computing infrastructure. *Proceedings of the International Conference on High Performance Computing and Networking Europe (HPCN Europe)*, Vienna, Austria, 28–30 April 1997. IEEE Computer Society Press: Los Alamitos, CA, 1997.
25. Haritsa J, Carey MJ, Livny M. Earliest deadline scheduling for real-time database systems. *Proceedings of the Real-Time Systems Symposium*, 1991.
26. Biyabani S, Stankovic J, Ramamritham K. The integration of deadline and criticalness in hard real-time scheduling. *Proceedings of the Real-Time Systems Symposium*, 6–8 December 1998.

27. Aydin H, Mejia-Alvarez P, Melhem R, Mosse D. Optimal reward-based scheduling of periodic real-time tasks. *Proceedings of the IEEE Real-Time Systems Symposium*, Phoenix, AZ, December 1999.
28. Sullivan WT III, Werthimer D, Bowyer S, Cobb J, Gedye D, Anderson D. A new major SETI project based on Project Serendip data and 100,000 personal computers. *Proceedings of the Fifth International Conference on Bioastronomy*, 1997. Available at: http://setiathome.ssl.berkeley.edu/.
29. Pande V. Protein folding on Internet-wide distributed computing (Folding@Home project). http://www.stanford.edu/group/pandegroup/Cosm/ [2001].
30. Blackstone. Post-genomic challenges drive life sciences to high throughput computing (HTC), *White paper*, April 2001. Available at: http://www.blackstonecomputing.com/products/whitePapers/.
31. CERN. The Large Hadron Collider (LHC) Computing Grid Project for high energy physics data analysis. http://lcg.web.cern.ch/LCG/ [May 2003].
32. Abramson D, Power K, Sosic R. Simulating computer networks using clusters of PCs. *HPC-TelePar'99 at the 1999 Advanced Simulation Technologies Conference (ASTC'99)*, San Diego, CA, 11–15 April 1999.
33. Smallen S, Casanova H, Berman F. Applying scheduling and tuning to on-line parallel tomography. *Proceedings of the IEEE/ACM Supercomputing (SC 2001) Conference*, Denver, CO, November 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
34. Casanova H, Bartol T, Stiles J, Berman F. Distributing MCell simulations on the Grid. *The International Journal of High Performance Computing and Supercomputing Applications* 2001; **15**(3).
35. Casanova H. SimGrid: A toolkit for the simulation of application scheduling. *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, May 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
36. Song H, Liu X, Jakobsen D, Bhagwan R, Zhang X, Taura K, Chien A. The MicroGrid: A scientific tool for modeling computational Grids. *Proceedings of Supercomputing (SC 2000) Conference*, Dallas, TX, November 2000. IEEE Press: Piscataway, NJ, 2000.
37. SPEC. SPEC CPU2000 results. http://www.specbench.org/osg/cpu2000/results/cpu2000.html [30 January 2002].
38. Buyya R, Abramson D, Giddy J, Stockinger H. Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation: Practice and Experience* 2002; **14**(13–15).
39. SC2002 Global Grid Testbed Collaboration. http://scb.ics.muni.cz/static/SC2002/.
40. Buyya R, Venugopal S. The Gridbus toolkit for service oriented grid and utility computing: An overview and status report. *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004)*, Seoul, Korea, 23 April 2004. IEEE Press: New Jersey, 2004; 19–36.
41. Yu J, Venugopal S, Buyya R. A market-oriented Grid directory service for publication and discovery of Grid service providers and their services. *Technical Report*, Department of Computer Science and Software Engineering, University of Melbourne, 2002.
42. Buyya R. The World-Wide Grid. http://www.gridbus.org/wwg/.