Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 194 (2009) 856–872

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

O.R. Applications

Maximizing business value by optimal assignment of jobs to resources in grid computing

Subodha Kumar [a,*], Kaushik Dutta [b,1], Vijay Mookerjee [c,2]

[a] *Information Systems and Operations Management, Michael G. Foster School of Business, University of Washington, Box 353200, Seattle, WA 98195, USA*
[b] *Decision Science and Information Systems, Florida International University, 11200 SW 8th Street, University Park Miami, FL 33199, USA*
[c] *Information Systems and Operations Management, School of Management, University of Texas, Dallas, Richardson, TX 75083-0688, USA*

Received 24 December 2006; accepted 17 December 2007
Available online 31 December 2007

Abstract

An important problem that arises in the area of grid computing is one of optimally assigning jobs to resources to achieve a business objective. In the grid computing area, however, such scheduling has mostly been done from the perspective of maximizing the utilization of resources. As this form of computing proliferates, the business aspects will become crucial for the overall success of the technology. Hence, we discuss the grid scheduling problem from a business perspective. We show that this problem is not only strongly NP-hard, but it is also non-approximable. Therefore, we propose heuristics for different variants of the problem and show that these heuristics provide near-optimal solution for a wide variety of problem instances. We show that the execution times of proposed heuristics are very low, and hence, they are suitable for solving problems in real-time. We also present several managerial implications and compare the performance of two widely used models in the real-time scheduling of grid computing.

Keywords: Evolutionary computations; Parallel computing; Scheduling; Heuristic; Integer programming

1. Introduction

Recent developments in processing power, data storage, and networking have made possible the migration of grid technology from the educational and research fields into the business realm (Berman et al., 2003; Glasgow, 2003). A grid is essentially a shared pool of networked resources that are modular, cost-effective, flexible, balanced, scalable, distributed, and standards-based (Castro-Leon and Munter, 2005; Intel, 2004a). In the grid computing, storage, data, and CPUs from multiple systems are connected into a managed and flexible computing environment (Intel, 2004b). Grid computing enables users to access idle or unutilized IT resources (De Roure et al., 2003). A grid becomes useful and meaningful when it both encompasses a large set of resources and serves a sizable community (Castro-Leon and Munter, 2005).

Grid computing has been put into use and achieved great success in the research arena. The most well-known example is the SETI@home project initiated by the University of California at Berkeley. However, the grid is not only of interest to the researchers, but its deployments are encompassing a broad swath of industry verticals that will take the grid well beyond its roots (Castro-Leon and Munter, 2005). Some of the successful examples of grids include Legion (Legion, 2007), NetSolve (Seymour et al., 2005), Nimrod/G (Buyya et al., 2000), and DISCWorld (Coddington, 2002). For a detailed tutorial on grid computing and its architecture and protocols, the reader is directed to Foster et al. (2002) and Castro-Leon and Munter (2005).

There are three sets of entities involved in the grid computing – job owners, resource owners, and the scheduler.

* Corresponding author. Tel.: +1 206 543 4777; fax: +1 206 543 3968.
E-mail addresses: subodha@u.washington.edu (S. Kumar), kaushik.dutta@fiu.edu (K. Dutta), vijaym@utdallas.edu (V. Mookerjee).
[1] Tel.: +1 305 348 3302; fax: +1 305 348 3497.
[2] Tel.: +1 972 883 4414; fax: +1 972 883 2089.

0377-2217/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2007.12.024

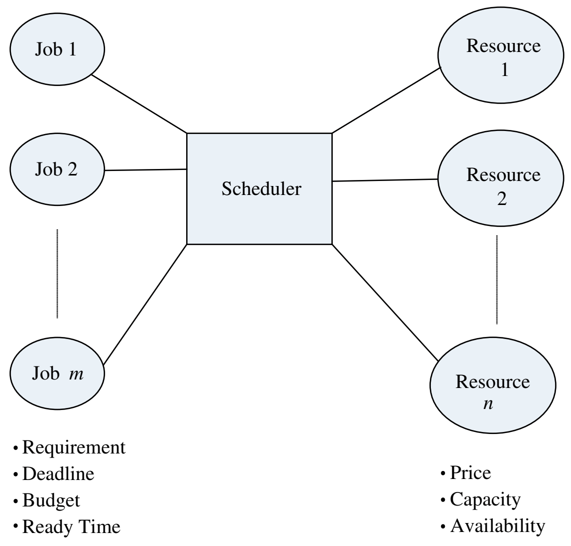

Fig. 1. Entities involved in grid computing and their parameters.

As illustrated in Fig. 1, both job owners and resource owners interact with the scheduler. Job owners specify their job requirements, the time by which the job must be finished (i.e., deadline), the budget available for their job, and the time at which the job will be ready to be processed (i.e., the ready time of the job) (Wolski et al., 2003). On the other hand, the resource owners set the price for using their resources, and specify their capacities and availabilities. In different settings of grid networks, the resource owners may decide the price using different mechanisms as explained by Wolski et al. (2003). Examples of resource pricing can be found at Sun Grid (Sun Grid, 2006; Utility Computing, 2006) and the cluster of Tsunamic Technologies Inc. (2006).

Many important research issues have arisen with the advent of grid computing. Some of these are in the area of security, compatibility, and functionality. However, the value of a grid system will be realized in enterprise systems only when the grid is used in an optimal way to serve a business objective (Cheliotis et al., 2005). An important issue in a grid is the optimal scheduling of jobs, i.e., given a set of resources and computing jobs, what is the best assignment of jobs to resources. There are two formal approaches to the allocation problem in grid: performance based, and economics based. The first is based on the assumption that the system response can be accurately modelled by various system parameters such as network usage, disk usage and the usage of CPU cycles. However due to the inherent nature of a grid, being composed of heterogeneous platforms distributed geographically across the world, developing such model remains elusive, making it difficult to define formally tractable performance based mechanisms (Wolski et al., 2003).

On the other hand, economics based approaches are attractive for several reasons. Firstly, the concept of *efficiency* or cost effectiveness is well defined; although it is different from the notion of efficiency typically understood in a computer performance evaluation setting. Secondly, economic systems and the assumptions upon which they are based seem familiar, making common intuition a more valuable research asset. Economics based approaches have garnered quite a bit of attention as a way to allocate grid resources (e.g., in the area of computational economy). As more grid systems will be deployed for commercial purposes, an economic approach to grid scheduling will become more appropriate (Cheliotis et al., 2005). Currently most economics based approaches to grid scheduling are studied using an auctions perspective (Schnizler et al., 2008). However, auctions based scheduling may not always be suitable for enterprise wide grid deployments.

For example, Merrill Lynch (ML) is working towards developing an enterprise wide grid that can be used by its various divisions to do complex financial computations (Private Communications). ML, one of the world's leading financial management and advisory companies, with offices in 36 countries and total client assets of approximately $1.6 trillion, requires complex, CPU intensive financial computations to allocate and manage their assets. In the proposed grid, various branches will be required to submit computational jobs to the centralized grid with a specification of the deadline, opportunity cost of not executing the job, budget, and estimated numbers of instructions required to finish the job. These jobs will then be allocated to appropriate resources meeting the constraints specified by the job owner. Following the spirit of a decentralized organizational structure, each of these branches will be required to pay to use the grid. In such scenarios, it is appealing to consider a schedule that optimizes an overall business objective of cost, rather than use auctions that essentially lead to competition between multiple divisions of the organization (Private Communications). Therefore, we study the problem of resource scheduling in grid computing networks with the goal of optimizing an economic objective, such as cost or value.

We use the real-world examples to illustrate the problems considered in this research. Let us first discuss the example of World Wide Grid (WWG) (World Wide Grid, 2007), which is represented in Fig. 2. World wide grid consists of computers in five continents and has been used for drug design (BioGrid, 2001). The economic model in WWG considers deadline and budget of jobs and dollar cost of resources to decide the assignment of jobs to resources. The few resources used in WWG are as given in Table 1 with their respective parameters. The details of existing resources in WWG are available at http://gridbus.cs.mu.oz.au/sc2003/list.html. Currently, WWG has 218 resources. The values of job parameters related to drug design case in WWG are shown in Table 2.

Recently, Buyya et al. (2005) proposed an economic approach for scheduling jobs in a grid. They considered budget, deadline, and processing time of jobs and the cost of cpu time of resources to develop Nimrod/G scheduler as

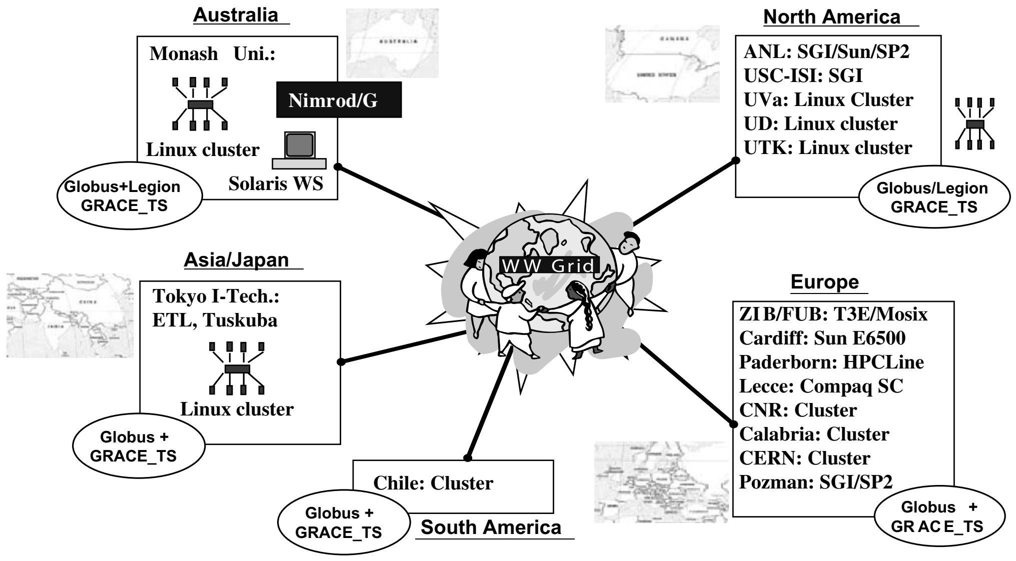

Fig. 2. World wide grid BioGrid, 2001.

Table 1
Sample resources in world wide grid (Buyya et al., 2005)

Resource hostname	Location	Specification	SPEC/MIPS rating	Price ($/time unit)
grendel.vpac.org	VPAC, Melb, Australia	Compaq, AlphaServer, CPU, OSF1, 4	515	8
hpc420.hpcc.jp	AIST, Tokyo, Japan	Sun, Ultra, Solaris, 4	377	4
hpc420-1.hpcc.jp	AIST, Tokyo, Japan	Sun, Ultra, Solaris, 4	377	3
hpc420-2.hpcc.jp	AIST, Tokyo, Japan	Sun, Ultra, Solaris, 2	377	3
barbera.cnuce.cnr.it	CNR, Pisa, Italy	Intel, Pentium/VC820, Linux, 2	380	1
onyx1.zib.de	ZIB, Berlin, Germany	SGI, Origin 3200, IRIX, 6	410	5
Onyx3.zib.de	ZIB, Berlin, Germany	SGI, Origin 3200, IRIX, 16	410	5
mat.ruk.cuni.cz	Charles U., Prague, Czech Republic	SGI, Origin 3200, IRIX, 16	410	4
marge.csm.port.ac.uk	Portsmouth, UK	Intel, Pentium/VC820, Linux, 2	380	1
green.cfs.ac.uk	Manchester, UK	SGI, Origin 3200, IRIX, 4	410	6
pitcairn.mcs.anl.gov	ANL, Chicago, USA	Sun, Ultra, Solaris, 8	377	3

Table 2
Parameters in world wide grid (BioGrid, 2001)

Deadline of job	2 hours
Budget	396,000 units
Number of jobs	165
Total cost incurred	115,200 units

Table 3
Parameters in Nimrod/G (Buyya et al., 2005)

Deadline of job	100–3600 time unit
Budget	5000–22,000 units
Number of jobs	200
Number of resources	10
Processing requirement of each job	10,000 machine instructions (MI)
Processing power of each resources	100 machine instructions per seconds (MIPS)
Cost of resources	1–8 unit per CPU time

part of the Globus framework. In this grid infrastructure, parameter values for jobs and resources were set as given in Table 3.

In both of the systems discussed above, the jobs are submitted to a logically centralized scheduler with detailed descriptions, called GridLet (Buyya et al., 2005), and the scheduler is responsible for allocating the jobs to the proper resource at the appropriate time so that a business objective is optimized. The business objective in such a scheduling mechanism varies from organization to organization such as least cost or least time (Buyya et al., 2005).

Similar to the examples presented above, we consider budget, deadline, and processing time of each job, and the price of each resource in our models. In addition, we also consider that there are limited resources at any instance. Hence, it may not be possible to assign each job to a resource. In this paper, we focus our attention towards the cost minimization problem for the sake of brevity. However, our model can be easily extended to support other performance or economic objectives, such as the maximization of revenue, or the minimization of total execution time.

We consider the following three problems that are usually encountered in the grid scheduling practice: (i) all tasks of a given job must be assigned to a single resource, (ii) different tasks of a job can be assigned to different resources, and (iii) jobs are scheduled in the real-time rather than in batches. Although it is common to assign all tasks of a job to a single resource in the grid, some grids allow the assignment of different tasks of the same job to different resources because of the associated benefits (Castro-Leon and Munter, 2005). We analyze both of these scenarios and examine the impact of distributed processing. Finally, we also study the real-time scheduling problem encountered in the grid computing.

We show that the grid scheduling problem considered in this research is not only strongly NP-hard, but it is also non-approximable, i.e., it cannot be approximated in polynomial time within arbitrarily good precision. Hence, we propose the heuristics to solve the problem. In order to study the effectiveness of the proposed heuristics, we first present the integer programming (IP) formulations. We solve these IP formulations to obtain the optimal solution or the lower bound on the optimal solution, which is used as a benchmark to examine the heuristics. We show that the proposed heuristics not only provide near-optimal solutions, but do so using very little CPU time. Hence, these heuristics are well suited for the scenarios where scheduling decisions must be made in real-time rather than in a batch environment.

In this research, we provide the insights and solutions for all three entities shown in Fig. 1. First, the proposed heuristics can be easily integrated with existing grid computing frameworks (such as Globus), and the scheduler can make assignment decisions automatically to serve a business objective. Second, we study the impact of parameters on the goals of job owners and resource owners. This study would help job owners and resource owners in deciding their parameter values, such as the budget for a job, price for a resource, etc. Finally, we analyze the performance of two models being used for real-time scheduling in grids – (i) a non-preemptive model in which a job must be completed after it is started, and (ii) a preemptive model in which a job can be stopped to start another job.

The rest of the paper is organized as follows. Section 2 provides the details of related work. In Section 3, we present the formal mathematical model. We examine the complexity of the problem and present our heuristic in Section 4. In this section, we also test the performance of the proposed heuristic. The results and managerial implications are provided in Section 5. In Section 6, we consider the variant of the problem where different tasks of a job can be assigned to different resources. Finally, a real-time scheduling problem is briefly analyzed in Section 7. Section 8 concludes the paper and provides directions for future research.

2. Related work

The most relevant works to our research are Buyya et al. (2002a,b, 2005). In these studies, as part of the World wide grid (World Wide Grid, 2007) and Nimrod/G (Buyya et al., 2000) scheduler framework, authors propose a greedy algorithm that meets the budget and deadline constraints of the jobs, and optimizes the cost and time. Although the authors have done an extensive job in addressing the importance of economic model in grid computing, providing implementation level details, and integrating economic scheduler with the existing grid framework Globus (Globus Alliance, 2007), these studies have several limitations. First, these scheduling algorithms assume that there are unlimited resources available all the time in the grid. In a real-world grid, however, the resources may join and leave at different times resulting in varying number of resources over time. Moreover, in none of the existing research, the theoretical analysis of such scheduling problem has been presented which is important to develop an effective heuristic that may provide a near-optimal solution. In order to bridge this gap, we develop the formal mathematical models for this important economic grid scheduling problem and propose heuristics that consider the limited resources and the limited availability of resources. We demonstrate that our heuristics provide the near-optimal solution for a wide variety of problem instances very quickly. Hence, our heuristics can be effectively used in the real-time setting and integrated with the existing grid schedulers. Additionally, we also analyze the impact of changing the parameter values on the solution.

Some other related works include He et al. (2003), which provides the scheduling algorithm for the grid computing that meets the QoS constraints of the jobs. Moreover, Angulo et al. (2002) propose a mechanism that schedules jobs to resources based on the processing requirements of the applications being executed. Finally, Min and Maheswaran (2002) present a scheduling algorithm that mixes the QoS and the priority of jobs to find the right resource for a job. The main objective of these algorithms is to satisfy the given requirements, rather than optimizing any business objective, such as cost or time. In contrast, we optimize the business objective while satisfying the constraints.

Another related research stream pertains to scheduling, i.e., the allocation of limited resources to optimize certain objective functions. In order to get more details about the scheduling literature, readers can refer to Herroelen et al. (1998), Lee et al. (1997) and Mokotoff (2001). Although the scheduling problems appear to be similar, there are stark differences in their solution methodologies. Most of these problems are NP-hard, and their heuristics are developed based on the specific problem domain knowledge. Hence, we cannot adopt any of these heuristics directly. In addition, due to the real-time behavior of grid computing framework, the scheduling problem in grid computing requires separate attention that needs to be solved effectively and efficiently, unlike most other job scheduling problems (such as airline or job-shop scheduling) where longer computational times may be afforded. To the best of our knowledge, the proposed methodology does not match with any of the existing solution techniques.

However, we adopt some ideas from the existing heuristics to design our algorithm.

3. Model description

In this section, we present the mathematical model of the problem. We begin with defining the parameters of the problem and the variables used in the model.

3.1. Parameters and variables

Assume that we have m jobs and n resources with a job i having e_i tasks. The known parameters about jobs and resources are as follows:

Exogenous parameters

t_{ijl}	total processing time required for the lth task of ith job if assigned to the jth resource
p_j	the price/unit time for the jth resource
A_j	time from which the jth resource is available
U_j	time from which the jth resource is unavailable
B_i	budget for the ith job
D_i	deadline of the ith job
R_i	ready time of the ith job
L_i	penalty cost of rejecting the ith job.

This variable defines the priority of jobs. A job i with the highest priority will have the highest penalty cost of rejection.

The processing time of a task on a resource (i.e., t_{ijl}) can be estimated using the existing techniques (Beltrame et al., 2001; Brandolese et al., 2001; Halang, 1989; Krishnaswamy et al., 2004; Yang et al., 1994) from the number of machine instructions (MI) required to execute a job. Before a job is deployed in the system, it is run through a set of staging systems. During the trial run in the staging environment, the MIs estimate for each task of a job can be derived. By combining this time estimate in MI with the processing power of each resource (machine instructions per seconds, MIPS) (SPEC, 2007), the scheduler can estimate the time required to run each task on each resource. This approach is already being used by an existing economic grid scheduler (Buyya et al., 2005).

Cheliotis et al. (2005) note that the price formation mechanisms are difficult to design in the grid computing. However, this issue has been discussed in several recent studies and the procedures are provided for the resource owners to set the prices (Bapna et al., 2008; Cheliotis et al., 2005; Wolski et al., 2003). It has also been highlighted that there is no need for the price formation mechanism to be visible to the job owners (Cheliotis et al., 2005). Hence, based on its needs, the owner of resource j can choose the price formation mechanism from the portfolio of procedures recommended in past studies. This mechanism can then be used by the owner of resource j to set p_j. However, some of the resource owners may have some flexibility in setting their price. Similarly, some of the job owners may also have some flexibility in setting their budget and deadline. Hence, we analyze the impact of changing each of these parameter values in Section 5 and discuss their managerial implications.

For each job, there is a certain business need and if the job is not processed by the job's deadline, a loss will be incurred. In a tightly constrained system, it may not be possible to assign all the jobs. In such cases, the scheduler need to schedule and run jobs so that the loss is minimized. Therefore, we introduce a penalty cost L_i for job i that is estimated loss that occurs from not processing the job by its deadline. The penalty cost may correspond to an opportunity cost or a direct cost that is incurred (e.g., a contractual loss as a result of a performance clause), and it is set for a job by the scheduler based on the relative costs associated with that job. This penalty cost will therefore create a priority among jobs, where the scheduler will intend to select jobs with the higher penalty values (i.e., the higher priority). Since the penalty cost is assigned by the scheduler based on the loss (e.g., contractual loss) associated with the rejection of the job, there is no issue of information asymmetry or artificially inflated priority in this model. The jobs coming from the same division of an organization may have some interdependency in the penalty costs. However, it can be captured up to some extent by assigning the penalty costs appropriately. Clearly, a more comprehensive model will have explicit dependencies in terms of the penalty costs. But, in practice, it will not only make the problem extremely difficult to solve, but would also require estimating a very large number of parameters. Next, we specify the decision variables of the model.

Variables

$x_{ijl} = 1$	if lth task of ith job is assigned to resource j; 0 otherwise $\forall i, j, l$
s_{il}	start time of the lth task of job i $\forall i, l$
$y_{ijkl} = 1$	if lth task of ith job is the kth assignment on resource j; 0 otherwise $\forall i, j, k, l$
f_{jk}	start time of the kth assignment on resource j if resource j has at least k tasks assigned $\forall j, k$

3.2. Cost minimization model

As discussed earlier, organizations (e.g., Merrill Lynch) would like to minimize the cost of allocating jobs to resources in their enterprise wide grid. In practice, some of the grids do not allow the allocation of different tasks of a job to different resources (Private Communications). In other words, all tasks of a job are assigned to the same resource in such grids as a single atomic unit. We first present a linear mixed integer programming (IP) model for the cost minimization problem in these grids. For notational convenience, we suppress the subscript for tasks in this model. The general model, where multiple tasks of a job are allowed to be assigned to difference resources, is considered later in Section 6

$$\text{Minimize} \quad \sum_i \sum_j t_{ij} p_j x_{ij} + \sum_i \left(1 - \sum_j x_{ij}\right) L_i, \tag{1}$$

$$\text{subject to :} \quad \sum_j t_{ij} p_j x_{ij} \leqslant B_i \quad \forall i, \tag{2}$$

$$s_i \geqslant A_j x_{ij} \quad \forall i, \tag{3}$$

$$s_i + \sum_j t_{ij} x_{ij} \leqslant \sum_j U_j x_{ij} \quad \forall i, \tag{4}$$

$$s_i + \sum_j t_{ij} x_{ij} \leqslant D_i \quad \forall i, \tag{5}$$

$$s_i \geqslant R_i \sum_j x_{ij} \quad \forall i, \tag{6}$$

$$\sum_j x_{ij} \leqslant 1 \quad \forall i, \tag{7}$$

$$\sum_i y_{ijk} \leqslant 1 \quad \forall j, k, \tag{8}$$

$$\sum_i y_{ijk} \geqslant \sum_i y_{ij(k+1)} \quad \forall j, k, \tag{9}$$

$$\sum_k y_{ijk} = x_{ij} \quad \forall i, j, \tag{10}$$

$$f_{j(k+1)} \geqslant f_{jk} + \sum_i t_{ij} y_{ijk} \quad \forall j, k, \tag{11}$$

$$s_i - f_{jk} - M_i^1 (1 - y_{ijk}) \leqslant 0 \quad \forall i, j, k, \tag{12}$$

$$s_i - f_{jk} + M_j^2 (1 - y_{ijk}) \geqslant 0 \quad \forall i, j, k, \tag{13}$$

$$x_{ij} \in \{0,1\} \quad y_{ijk} \in \{0,1\} \quad f_{jk} \geqslant 0$$

$$s_i \geqslant 0, \tag{14}$$

where

$$M_i^1 = \max_j\{\min\{U_j, D_i\} - t_{ij}\} \quad \text{and}$$
$$M_j^2 = \min\{U_j, \max_i\{D_i\}\}. \tag{15}$$

The first term in Eq. (1) represents the cost of all the assigned jobs and the second term represents the penalty for all rejected jobs. The constraint set (2) is the budget constraint for each job. The constraint set (3) ensures that the processing of a job on a resource starts only after that resource is available. Similarly, the constraint set (4) ensures that the processing of a job on a resource ends before that resource becomes unavailable. The deadline constraint for each job is modeled in the constraint set (5). The constraint set (6) ensures that the processing of a job starts only after it is ready to be processed. The constraint set (7) makes sure that a job is assigned to only one resource. Similarly, the constraint set (8) ensures that there is at most one job at the kth assignment of resource j. The constraint set (9) guarantees that there must be a job at the kth assignment of a resource before the $(k+1)$th assignment of that resource has a job. The constraint set (10) ensures that if a job is assigned to a resource it must be assigned at some kth assignment, otherwise the job should not be assigned at any assignment of that resource. The constraint set (11) represents the constraints for variables f_{jk}. Finally, the constraint sets (12) and (13) capture the definitions of variables y_{ijk} and f_{jk} as follows: When $y_{ijk} = 1$, then the ith job is the kth assignment on resource j, and therefore, from (12) and (13), $f_{jk} = s_i$. On the other hand, when $y_{ijk} = 0$, then, from (14) and (15), it is easy to show that both (12) and (13) are redundant.

In a performance-based model, the constraints will remain exactly the same as the cost minimization model. However, the objective function will be different. For example, in the time minimization model, the objective function will be $\sum_i \sum_j t_{ij} x_{ij} +$ penalty for rejected jobs. The penalty will be similar to that in Eq. (1), but it will be represented in terms of time. The main difference between the objective functions of time minimization model and the cost minimization model is that the former does not consider price. If the business objective is cost minimization, then the time minimization model may result in a sub-optimal solution. The key difference between the solutions of these two models lies in the fact that the time minimization model assigns jobs to the resources that take least amount of time, without considering the price of those resources. Hence, the time minimization model may result in a solution where the cost is considerably higher than the optimal cost.

4. Analysis of the cost minimization model

In this section, we first present the complexity of the problem. We show that the problem is strongly NP-hard, and hence it cannot be solved optimally in a reasonable amount of time (Garey and Johnson, 1979). Then, we present some interesting theoretical results regarding the *polynomial time approximation scheme*. Finally, we propose an efficient heuristic and demonstrate its performance by comparing its solution with a bound on the optimal solution.

4.1. Complexity of the problem

Theorem 1. *The cost minimization problem is strongly NP-hard.*

Proof. Let us first consider the decision problem corresponding to the cost minimization problem as follows.

Decision problem: Given the set of m jobs and n resources, and the values of t_{ij}, p_j, A_j, U_j, B_i, D_i, R_i, and L_i for each job i and each resource j, does there exist a valid assignment of the jobs for which the objective function value of the cost minimization model is less than or equal to ϕ? Here, ϕ is some pre-specified value.

First, note that a positive answer to the decision problem is verifiable in time O(m) and hence, it is in NP (Garey and Johnson, 1979). We now show that the well-known strongly NP-complete problem 3-*Partition* reduces to the decision problem given above.

3-*Partition problem*: Given positive integers α, λ, and a set of integers $X = \{x_1, x_2, \ldots, x_{3\alpha}\}$ with $\sum_{k=1}^{3\alpha} x_k = \alpha\lambda$ and $\lambda/4 < x_k < \lambda/2 \ \forall k$, does there exist a partition $\{\Omega_1, \Omega_2, \ldots, \Omega_\alpha\}$ of X with $|\Omega_\ell| = 3$ and $\sum_{x_i \in \Omega_\ell} x_i = \lambda \ \forall \ell$?

Given an arbitrary instance of the 3-Partition problem, an instance of the cost minimization problem is constructed as follows: We first set $m = 3\alpha$, $n = \alpha$, and $t_{ij} = x_i$ for $i = 1, 2, \ldots, 3\alpha$; $j = 1, 2, \ldots, \alpha$. Now, $p_j = 1$, $A_j = 0$, and $U_j = \lambda\ \forall j$. For each job i, $B_i \geqslant \lambda/2$, $D_i \geqslant \lambda$, $R_i = 0$, and $L_i > \alpha\lambda$. Finally, $\phi = \alpha\lambda$. If any job is rejected, then the objective function value will exceed ϕ, because $L_i > \alpha\lambda\ \forall i$. Moreover, since $\lambda/4 < t_{ij} < \lambda/2$ for each job i and resource j and $U_j = \lambda\ \forall j$, we cannot assign more than 3 jobs to any resource. Therefore, each resource should have exactly 3 jobs, because there are 3α jobs and α resources. A 3-Partition problem then provides a solution to the decision problem. Conversely, a solution to the decision problem with $\phi = \alpha\lambda$ provides a solution to the 3-Partition problem. □

From the proof of Theorem 1, the following corollary directly follows.

Corollary 1. *The cost minimization problem is strongly NP-hard even when all the resources have same price, same available time, and same unavailable time.*

4.2. Polynomial time approximation scheme

A polynomial-time approximation scheme (PTAS) for an optimization problem is a family of $(1+\epsilon)$-approximation algorithms for all $\epsilon > 0$ that operate in the time bounded by a polynomial function of the length of the input (Garey and Johnson, 1978). Moreover, a PTAS is also the fully polynomial time approximation scheme (FPTAS) if its time complexity is bounded by a polynomial function of both the length of the input and $1/\epsilon$ (Garey and Johnson, 1979).

Since the cost minimization problem is strongly NP-hard, it is not possible to obtain a FPTAS (Garey and Johnson, 1978). Moreover, we obtain the following non-approximability result.

Theorem 2. *The cost minimization problem is MAX SNP-hard and thus does not have a polynomial time approximation scheme. In other words, the cost minimization problem cannot be approximated in polynomial time within arbitrarily good precision.*

Proof. Let us consider an instance of the cost minimization problem where $p_j = 1$, $A_j = 0$, and $U_j = \max_i R_i + \sum_i t_{ij}$ for each resource j, and $B_i = \max_j t_{ij}$, $D_i = \max_i R_i + \sum_i t_{ij}$, and $L_i = \max_j t_{ij} + 1$ for each job i. Now, we show that this instance of the cost minimization problem, say instance I, is equivalent to an arbitrary instance of the problem $R|r_i|\sum C_i$, which is defined below.

Problem $R|r_i|\sum C_i$: There are m independent jobs, $i = 1, 2, \ldots, m$ that have to be scheduled on exactly one of the n machines. Every machine can process at most one job at a time. If job i is scheduled on machine j, then the processing time required is t_{ij}. Every job i has a release time $r_i (= R_i)$ at which it becomes available for processing. The objective is to minimize the total job completion time.

In instance I, any resource can process all the jobs because each resource is always available. Similarly, any job can always be finished before its deadline. Moreover, each job can be assigned to any resource because its budget is more than or equal to the cost of processing on any resource. Finally, each job must be assigned because the penalty cost of rejecting a job is always higher than its cost of processing on any resource. Once we ensure that each job must be assigned and it can be assigned to any resource, the cost minimization problem and the problem $R|r_i|\sum C_i$ are equivalent.

The concept of MAX SNP-hardness is introduced by Papadimitriou and Yannakakis (1991); the readers can refer to Papadimitriou and Yannakakis (1991) for an extensive explanation of MAX SNP-hardness. Hoogeveen et al. (2001) show that the problem $R|r_i|\sum C_i$, referred to as the *unrelated parallel machine scheduling problem*, is MAX SNP-hard. Therefore, the cost minimization problem is also MAX SNP-hard based on the equivalence shown above. Moreover, Arora et al. (1992) show that if there exists a PTAS for some MAX SNP-hard problem, then P = NP. Hence, the cost minimization problem does not have a PTAS. □

Based on the proof of Theorem 2, it is straightforward to obtain the following corollary.

Corollary 2. *The cost minimization problem does not have a polynomial time approximation scheme even when all the resources have same price, same available time, and same unavailable time, and all the jobs have same budget, same deadline, and same penalty cost.*

Now we present some interesting approximation results for the following three special cases of the cost minimization problem.

1. We begin with the NP-hard problem $R|r_i|\sum C_i$, which is defined earlier. As explained in the proof of Theorem 2, problem $R|r_i|\sum C_i$ is a special case of the cost minimization problem where $p_j = 1$, $A_j = 0$, and $U_j = \max_i R_i + \sum_i t_{ij}$ for each resource j, and $B_i = \max_j t_{ij}$, $D_i = \max_i R_i + \sum_i t_{ij}$, and $L_i = \max_j t_{ij} + 1$ for each job i. For problem $R|r_i|\sum C_i$, Hall et al. (1997) present a $\frac{16}{3}$-approximation algorithm. Hence, we also have a $\frac{16}{3}$-approximation algorithm for the special case of the cost minimization problem.
2. Now we consider a more specialized version of the cost minimization problem, where we have $t_{ij} = t_i$ for each job i and each resource j, in addition to the values considered in the previous special case. This special case is equivalent to the problem of minimizing completion time in identical parallel machine scheduling with release dates, denoted by $P|r_i|\sum C_i$, which is also a NP-hard problem. For this problem, Chekuri et al. (2001) provide a 2.83-approximation algorithm. Therefore, we have 2.83-approximation algorithm for this special case.

3. Finally, we consider the most simplified version of the cost minimization problem, where $R_i = 0$ for each job i in addition to the values considered in the previous special case. This version is equivalent to the problem of minimizing completion time in identical parallel machine scheduling, denoted by $P\|\sum C_i$. Since the *Shortest Processing Time first* (*SPT*) rule is optimal for $P\|\sum C_i$ (Pinedo, 1995, p. 79), it is also optimal for this special case of the cost minimization problem.

4.3. Optimal solution

In order to obtain the optimal solution, we first try to solve the IP formulation given in Section 3 using CPLEX 8.1.0. CPLEX can solve small problem instances (up to 10 jobs and 5 resources) to optimality in a reasonable amount of time. However, as the number of jobs and resources increases, CPLEX fails to provide optimal solution. Hence, we propose an efficient heuristic for the cost minimization model, called the *Highest Rank Earliest Deadline* (HRED).

4.4. Highest rank earliest deadline (HRED) heuristic

In the HRED heuristic, we try to strike a balance between the objective function and constraints in such a way that the total cost of assigning jobs to resources is reduced. The HRED heuristic first calculates the rank (V_{ij}) for each combination of assigning job i to resource j, which is defined as follows: $V_{ij} = (L_i - t_{ij}p_j)$. Clearly, jobs with higher penalty cost will have a higher rank. The heuristic sorts the combinations of jobs and resources in the non-increasing order of their ranks. Jobs are then assigned from this sorted list.

It is easy to see that there should not be any such assignment for which the cost of assignment is more than either the penalty cost, or the budget constraint. Therefore, such combinations of jobs and resources are first removed from the sorted list. Now, we start from the top of the updated sorted list. Let the job be β and resource be γ for the first combination in the list. If it is feasible to assign job β to resource γ based on the values of $t_{\beta\gamma}, p_\gamma, A_\gamma, U_\gamma, B_\beta, D_\beta$, and R_β, then β is assigned to γ, and all the combinations of β are removed from the sorted list. This process is repeated for the next combination in the updated list and so on. The heuristic keeps traversing the list until the list is empty. The formal algorithm is now given below.

Algorithm HRED

Step 1: Calculate the rank $V_{ij} = (L_i - t_{ij}p_j)$ for each combination of job i and resource j, and sort them in non-increasing order of V_{ij}. Here, $i = 1, 2, \ldots, m$; and $j = 1, 2, \ldots, n$.

Step 2: Remove all those combinations from the sorted list where V_{ij} is negative or $t_{ij}p_j > B_i$.

Step 3: Let β' be the job and γ' be the resource in the first combination of the updated sorted list, and η be the set of jobs that have already been assigned to γ'.

Step 4: Combine all the jobs in set η with the job β' and sort all of them in non-decreasing order of their deadline and ready time. Now try to assign these jobs to resource γ' in the sorted order. If all of these jobs can be assigned without violating the deadline constraint of any of the jobs or the availability constraint of resource γ', then the combination of β' and γ' is considered to satisfy the time constraint.

Step 5: If the time constraint is satisfied for the combination of β' and γ', then assign β' to γ' and remove all the combinations of β' from the sorted list.

Step 6: If the updated list is empty, then terminate; otherwise go to *Step 3*.

The time complexity of algorithm HRED lies in the complexity class $O(mn\log(mn) + m^2n\log m + mn^2)$.

4.5. Performance of HRED heuristic

As CPLEX was unable to provide any feasible solution for the practical size problems in a reasonable amount of time, we took a two stage approach to judge the performance of HRED heuristic. In the first stage, we compare the HRED solution with the CPLEX optimal solution for small size problems (up to 10 resources and 10 jobs). In the next stage, we compare the solution of HRED heuristic with the lower bound obtained by solving the linear programming (LP) relaxation of the IP formulation given by Eqs. (1)–(15). For all these experiments, HRED is coded in C++ on a Pentium 4 computer (2.4 GHz, 2 GB RAM) with Windows XP as the operating system.

4.5.1. Performance for small size problems

The size of this set of problems was chosen based on what can be solved optimally by CPLEX. We choose two values each for the number of jobs (5 and 10) and the number of resources (5 and 10). Hence, there are total four (2×2) combinations. For each of these combinations, we generate four problem instances by choosing two values each for the range of B_i (20–30 and 90–100) and the range of U_j (40–50 and 80–100). Thus, we have total sixteen (4×4) problem instances. The ranges for other parameter values are as follows: p_j–(4–8), D_i–(80–100), and A_j–(0–30). The actual values of these parameters are drawn randomly from their ranges.

In Table 4, we first report the percentage gap of the HRED cost from the optimal cost which is calculated as follows:

$$\text{Percentage Gap of HRED Cost from Optimal Cost} = \frac{(\text{HRED Cost}) - (\text{Optimal Cost})}{\text{Optimal Cost}} \times 100.$$

Table 4
Performance of the HRED heuristic for small size problems

Prob #	Number of jobs	Number of resources	Range of budget	Range of U_j	Percentage gap of HRED cost from optimal cost	Percentage gap of LP bound from optimal cost	Number of jobs rejected
1	5	5	20–30	40–50	0	23.38	3
2				80–100	0	34.29	3
3			90–100	40–50	0	8.33	1
4				80–100	0	0	0
5	5	10	20–30	40–50	0	22.10	1
6				80–100	0	22.03	1
7			90–100	40–50	0	1.00	0
8				80–100	0	0	0
9	10	5	20–30	40–50	0	14.40	5
10				80–100	0	42.95	4
11			90–100	40–50	0	0	3
12				80–100	0	0	0
13	10	10	20–30	40–50	3.19	34.13	7
14				80–100	0	45.82	3
15			90–100	40–50	0	0	1
16				80–100	0	0	0

Similarly, we calculate and report the percentage gap of the cost obtained by solving the LP relaxation of the problem from the optimal solution. The purpose of reporting this gap is to study the quality of the lower bound provided by LP relaxation. It is an important study for our next set of experiments with large size problems, where we measure the performance of HRED heuristic by comparing its solution with the lower bound obtained by LP relaxation.

After several iterations, we find that the following two additional constraints improve the bound provided by LP relaxation. Note that these constraints do not affect the optimal solution

$$f_{jk} \geqslant A_j \forall j,k, \quad \text{and} \quad f_{jk} \leqslant U_j \forall j,k.$$

As can be seen in Table 4, the HRED heuristic provides the optimum solution for all the problem instances except problem# 13. We also note that the gap between the lower bound provided by LP relaxation and the optimal solution is very large when the number of jobs rejected in the optimal solution (shown in the last column of Table 4) is high. This is due to the fact that the LP relaxation can divide a job and partially allocate it to different resources, whereas such partial allocation is not possible in the optimal solution.

4.5.2. Performance for large size problems

In this section, we report the performance of the HRED heuristic by comparing its solution with the lower bound obtained by LP relaxation. The test bed is chosen to balance the goals of realism and feasibility. Realism is the goal of choosing problem instances that are representative of problems encountered in practice (based on our private communications and some open grid systems such as GridBus). Feasibility is the goal of being able to solve the LP relaxation in a reasonable amount of time so that we can test the performance of HRED. Following this, we consider problem sizes up to 100 jobs and 70 resources. Although HRED can quickly solve much larger problem sizes, we limit ourselves to 100 jobs and 70 resources because CPLEX is unable to solve even the LP relaxation for larger problem sizes.

In this experiment, we consider four values each for the number of jobs (25, 50, 75 and 100) and the number of resources (10, 30, 50 and 70). Hence, there are sixteen (4×4) problem classes. For each problem class, we choose two values each for the ranges of p_j (5–10 and 15–20), U_j (70–100 and 50–90), B_i (100–150 and 120–180), and D_i (60–90 and 80–100). Hence, there are sixteen ($2 \times 2 \times 2 \times 2$) problem instances for each problem class. The ranges for other parameters are as follows: t_{ij}–(2–20), A_j–(0–30), and L_i–(100–150). The actual parameter values are drawn from these ranges randomly.

For the IP formulation of these problems, CPLEX was unable to provide any bound in a reasonable amount of time. Therefore, we determine the percentage gap of the HRED solution from the lower bound provided by LP relaxation as follows:

$$\text{Percentage Gap of HRED Cost from LP Bound} = \frac{(\text{HRED Cost}) - (\text{LP Bound})}{\text{LP Bound}} \times 100. \tag{16}$$

The LP relaxation is also solved using CPLEX 8.1.0. Each row in Table 5 shows the result for one problem class. For each problem class, the *maximum*, *average* and *minimum* percentage gaps are reported over sixteen problem instances in that class.

The computational results in Table 5 indicate that the average percentage gap of the HRED solution from the LP bound varies between 0% and 1.5% barring just one problem class (i.e., the problem class 13). We find that

Table 5
Performance of the HRED heuristic for large size problems

Problem class	Number of jobs	Number of resources	Percentage gap of HRED cost from LP bound		
			Maximum	Average	Minimum
1	25	10	3.66	0.67	0
2		30	1.11	0.07	0
3		50	0	0	0
4		70	0.72	0.08	0
5	50	10	5.91	1.02	0
6		30	0.30	0	0
7		50	0	0	0
8		70	0	0	0
9	75	10	4.13	1.50	0
10		30	0.74	0.05	0
11		50	0.08	0.01	0
12		70	0.38	0.02	0
13	100	10	24.51	6.02	0
14		30	0.58	0.09	0
15		50	0	0	0
16		70	0.15	0.01	0

the number of jobs rejected in the solutions of problem instances for problem class 13 is very high. Hence, based on the results of Section 4.5.1, the average high gap for problem class 13 can be explained by the high gap between the optimal solution and the LP bound. Table 5 also shows that the minimum percentage gap for every problem class is 0, which indicates that the HRED heuristic obtains the optimal solution for at least one problem instance in every class, for which the LP bound is also same as the optimal solution.

The maximum average CPU time for running the HRED heuristic is about 0.004 second. Moreover, the CPU time for many problem instances is zero. It means that the CPU time is less than one millisecond for those problem instances. Also, for larger size problems with 100 resources and 1000 jobs, HRED provides the result within less than 30 second. This result is very promising, because the scheduling heuristic needs to run very quickly in a grid computing environment. It also makes the HRED algorithm suitable for real-time scheduling, as explained in Section 7.

5. Discussions and managerial implications

In this section, we present the sensitivity of HRED solution with various parameters, and provide managerial insights. Based on our discussion in Section 1 and Fig. 1, we consider two parameters associated with the job owners: deadline and budget, and two parameters associated with the resource owners: price and availability.

5.1. Sensitivity to the deadline of jobs

In Fig. 3a, we change the range from which the deadline of each job is drawn randomly, and fix the ranges for all other parameter values as $p_j \in [7–13]$, $U_j \in [20–30]$, $D_i \in [0–100]$, $A_j \in [0–20]$, $R_i \in [0–10]$, $L_i \in [100–150]$, $B_i \in [1000–2000]$, and $t_{ijl} \in [2–17]$. As expected, the number of jobs rejected is very high when the value of deadline is small. Therefore, the cost is very high for smaller values of deadline due to penalty, and it decreases sharply with increase in the deadline until the number of rejections becomes zero. After the number of rejections becomes zero, the cost still decreases with increase in the deadline because the jobs can be assigned to less costly resources by increasing the deadline. However, it remains steady after the deadline reaches a certain threshold. Hence, there is no advantage for the job owners to increase the deadline beyond this threshold. This is an important insight for job owners because the opportunity cost can be expected to be convex (or at best linear) in the deadline.

In Fig. 3a, we also plot the total revenue of all resource owners. As the number of rejections decreases, the revenue of resource owners increases because more jobs are assigned to resources. However, this revenue starts decreasing when the number of rejections becomes zero because the jobs can be assigned to less costly resources as the deadline increases. Clearly, the revenue of resources owners is same as the cost when the number of rejections is zero, and hence, it becomes steady after a threshold.

Both the cost and revenue in Fig. 3a are normalized to plot them along with the number of rejections in the same graph. Fig. 3b illustrates the change in the cost of a single job owner with increase in its deadline for four different values of the number of resources. In this figure, we show

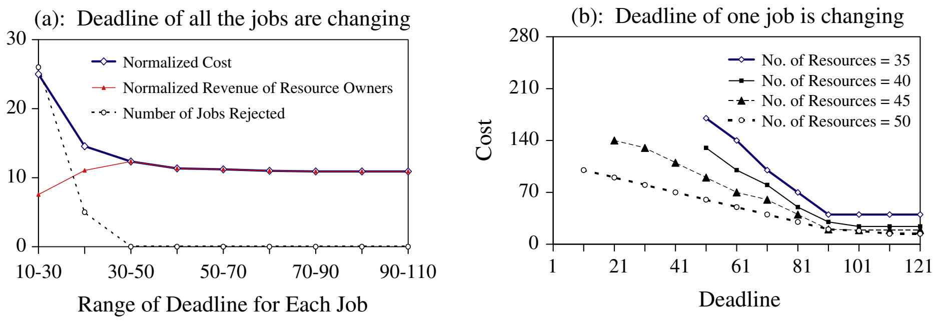

Fig. 3. Sensitivity to the deadline of jobs. (a) Range of deadline for each job. (b) Deadline.

that the cost first decreases with increase in the deadline and it becomes steady after a threshold value of deadline. Although the threshold values are similar for the four curves shown in Fig. 3b, they are not the same and may be very different for the other systems. Hence, the job owners need to decide the deadline based on the state of the system (such as the number of resources). Such aggregate information can be easily provided to the job owner by the grid scheduler to facilitate the best use of the grid.

We also observe in Fig. 3b that the cost decreases as the number of resources increases. When the number of resources is higher, the jobs have more options for the assignment, and hence can be assigned to the less costly resource. In Fig. 3b, we do not report the cost when the job is rejected. Hence, there are few missing points in Fig. 3b for some lower values of deadline.

5.2. Sensitivity to the budget of job owners

Fig. 4a plots the normalized cost and the number of rejections for different ranges of the job owners' budget when the other parameter values are fixed within the following ranges: $p_j \in [10–50]$, $D_i \in [40–50]$, $L_i \in [100–150]$, $t_{ijl} \in [15–20]$, $A_j \in [0–20]$, $U_j \in [20–30]$, and $R_i \in [0–10]$. When the budget for all the jobs increases, the set of accepted jobs changes and the total cost decreases. It may look counter-intuitive, but we observe by closer examination that the higher budget increases the solution space, and hence provides better solutions. However, the cost does not decrease with an increase in the budget after all the jobs are accepted (i.e., the number of jobs rejected is zero), because the jobs are always assigned with the objective of minimizing the cost. This phenomenon is further explained below.

In Fig. 4b, the budget of one job is changing and everything else is fixed. We do not report the cost when that job is rejected. Hence, there are few missing points in the plot. We can see that the job is accepted when the budget reaches a threshold. Moreover, this threshold depends on the state of the system (i.e., the number of resources). After the threshold, the cost of the job does not change. Again, since the job is always assigned with the objective of minimizing the cost, the improvement is not possible by increasing the budget after it is accepted. Therefore, the job owner should keep the budget at the level where it can be accepted, but he does not gain any advantage by increasing the budget beyond that level.

Fig. 4b also shows that the cost decreases as the number of resources increases. When there are more resources, the job has more options for the assignment, and hence the cost of assignment decreases. Also, as the number of resources increases, the job gets accepted at the lower value of budget.

5.3. Sensitivity to the price/unit time of resources

In Fig. 5a, we vary the range for the price/unit time of each resource, whereas the other parameter values are chosen randomly from the following ranges: $B_i \in [1000–2000]$, $D_i \in [40–50]$, $L_i \in [100–150]$, $t_{ijl} \in [15–20]$, $A_j \in [0–20]$, $U_j \in [20–30]$, and $R_i \in [0–10]$. Clearly, the cost increases as the price of all the resources increases. With an increase in the price of resources, the number of rejections initially stays at zero, but becomes positive and starts increasing after a certain threshold. As expected, the revenue of resource owners increases when the price increases and the number of rejections stays at zero. An interesting result observed in Fig. 5a is that the revenue still increases with the increase in the price when the number of rejections become positive. In this region, the increase in price overcompensates for the decrease in revenue due to rejections. However, when the number of rejections increases beyond a limit, the revenue starts decreasing with increase in the price. In this region, the decrease in revenue due to rejections dominates the increase in revenue due to increase in price.

The above discussion indicates that there is an optimal price level for the resource owners. The revenue of resource owners decreases by both increasing or decreasing the price from that level. Moreover, this level depends on the state of the system. This phenomenon is explained by Fig. 5b. In this figure, the price of one resource is changing and everything else is fixed. As can be seen in the figure, revenue of the resource owner (for which the price is changing) first increases and then decreases (similar to Fig. 5a). As the number of jobs increases, more job owners are willing to pay high price, and therefore the optimal level increases.

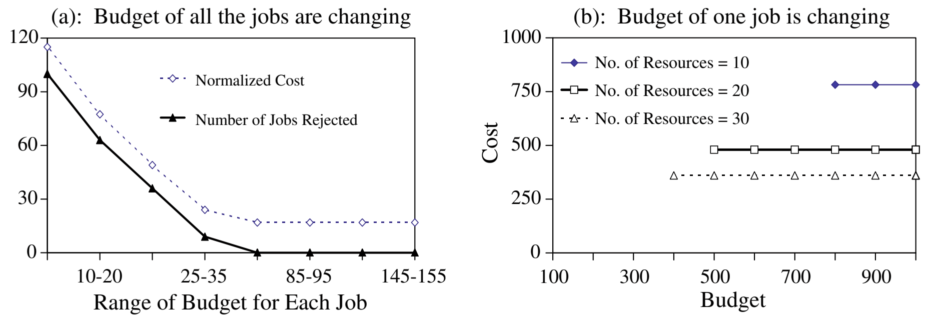

Fig. 4. Sensitivity to the budget of job owners. (a) Range of budget for each job. (b) Budget.

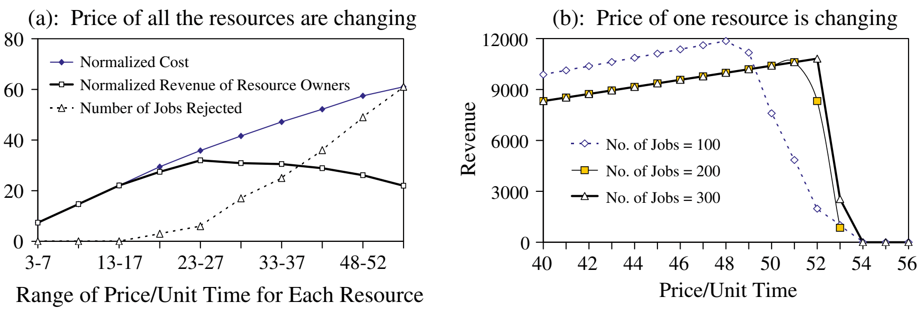

Fig. 5. Sensitivity to the price/unit time of resources. (a) Range of price/unit time for each resource. (b) Price/unit time.

Hence, the resource owner needs to optimize the price level based on the state of the system.

5.4. Sensitivity to the availability of resources

In order to study the sensitivity to the availability of resources, we fix A_j to zero for all j and change the value of U_j as the other parameter values are chosen randomly from the following ranges: $B_i \in [1000–2000]$, $D_i \in [40–50]$, $L_i \in [100–150]$, $t_{ijl} \in [15–20]$, $p_j \in [2–20]$, and $R_i \in [0–10]$. Note that an increase in U_j implies that the resource j is available for longer time.

Clearly the number of rejections decreases initially with an increase in the availability of resources (see Fig. 6a). During that period, the cost decreases sharply because of a reduction in the penalty cost. Moreover, the revenue of resources owners increases sharply in that period because more jobs are assigned to resources. Once all the jobs are accepted, increasing the availability of all resources slowly decreases both the cost and the overall revenue of all resource owners. It is an interesting result which shows that if all the resource owners increase their availability beyond a certain level, then they are worse off as a whole in terms of the total revenue.

However, Fig. 6b shows that the revenue of an individual resource owner either increases or remains constant when the availability is increased keeping everything else fixed. Hence, although it is worse for all the resource owners to increase their availability simultaneously, but it is advantageous for one resource owner to increase availability when the parameter values for other resources owners are fixed. As can be seen in the figure, the increase in revenue is a step function which becomes constant after certain level of U_j. Therefore, a resource owner should increase availability based on the number of jobs in the system and the action of other resource owners.

6. Distributed processing of tasks in a job

Next, we consider a general model where the multiple tasks of a job are allowed to be assigned to different resources. Typically in a grid environment, a job cannot be partially done, i.e., either all the tasks of a job is completed, or nothing gets done and the job gets rejected. Moreover, an additional communication cost is incurred when different tasks of a job are assigned to different resources (Li, 2005; Ernemann et al., 2002). Therefore, we need to introduce few additional variables which are defined as follows:*Additional variables*

$z_i = 1$	is ith job is accepted; 0 otherwise $\forall i$
$w_{ij} = 1$	if at least one task of ith job is assigned to resource j; 0 otherwise $\forall i, j$.

Clearly, there is no communication cost when all the tasks of a job are assigned to one resource. Based on the communication cost model proposed by Hamscher et al. (2000), the communication cost for a job in our model is proportional to the number of additional resources to which it is assigned. From the definition, $\sum_j w_{ij}$ is the total number of resources to which the tasks of job i are assigned. Therefore, for a given constant C, the communication cost for job i is given by $C\left(\sum_j w_{ij} - 1\right)$ if the job i is

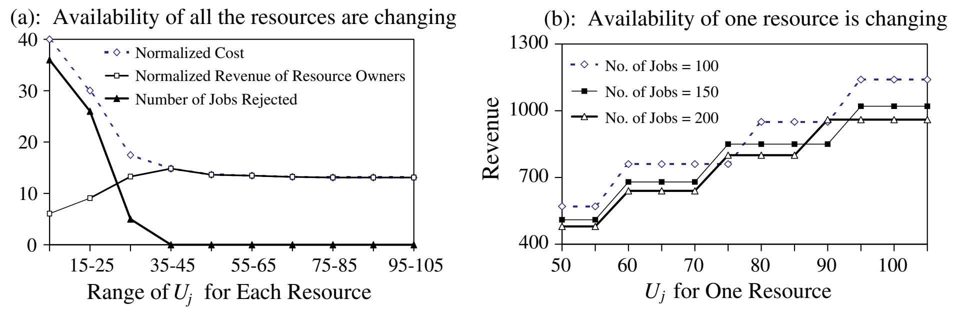

Fig. 6. Sensitivity to the availability of resources. (a) Range of U_j for each resource. (b) U_j for one resource.

assigned (i.e., $z_i = 1$). If the job i is not assigned, then $\sum_j w_{ij} = 0$. Hence, we can write the total communication cost as $C\sum_i(\sum_j w_{ij} - 1) + C\sum_i(1 - z_i)$.

The formal IP formulation for the general model is now given below

$$\text{Minimize} \quad \sum_i \sum_j \sum_l t_{ijl} p_j x_{ijl} + \sum_i \left(1 - \sum_j x_{ij}\right) L_i + C\sum_i \left(\sum_j w_{ij} - 1\right) + C\sum_i (1 - z_i), \tag{17}$$

$$\text{subject to}: \quad \sum_j \sum_l t_{ijl} p_j x_{ijl} + C\left(\sum_j w_{ij} - 1\right) + C(1 - z_i) \leqslant B_i \quad \forall i, \tag{18}$$

$$s_{il} \geqslant A_j x_{ijl} \quad \forall i, l,$$

$$s_{il} + \sum_j t_{ijl} x_{ijl} \leqslant \sum_j U_j x_{ijl} \quad \forall i, l,$$

$$s_{il} + \sum_j t_{ijl} x_{ijl} \leqslant D_i \quad \forall i, l,$$

$$s_{il} \geqslant R_i \sum_j x_{ijl} \quad \forall i, l,$$

$$\sum_j x_{ijl} \leqslant 1 \quad \forall i, l,$$

$$\sum_i \sum_l y_{ijkl} \leqslant 1 \quad \forall j, k,$$

$$\sum_i \sum_l y_{ijkl} \geqslant \sum_i \sum_l y_{ij(k+1)l} \quad \forall j, k,$$

$$\sum_k y_{ijkl} = x_{ijl} \quad \forall i, j, l,$$

$$f_{j(k+1)} \geqslant f_{jk} + \sum_i \sum_l t_{ijl} y_{ijkl} \quad \forall j, k,$$

$$s_{il} - f_{jk} - M^1_{il}(1 - y_{ijkl}) \leqslant 0 \quad \forall i, j, k, l,$$

$$s_{il} - f_{jk} + M^2_j(1 - y_{ijkl}) \geqslant 0 \quad \forall i, j, k, l,$$

$$\sum_j \sum_l x_{ijl} = e_i z_i \qquad \forall i, \tag{19}$$

$$w_{ij} \geqslant x_{ijl} \quad \forall i, j, l, \tag{20}$$

$$x_{ijl} \in \{0,1\} \quad y_{ijkl} \in \{0,1\} \quad z_i \in \{0,1\}$$

$$w_{ij} \geqslant 0 \quad f_{jk} \geqslant 0 \quad s_{il} \geqslant 0,$$

where

$M^1_{il} = \max_j\{\min\{U_j - t_{ijl}, D_i - t_{ijl}\}\}$ and

$M^2_j = \min\{U_j, \max_i\{D_i\}\}$.

Most of the terms in the above model follow directly from the previous model. Here, we explain only the additional components. The additional two terms in the objective function (17) incorporates the integration cost, which is explained earlier. The budget constraint set (18) also includes the same integration cost. The constraint set (19) ensures that either all the tasks of a job are assigned or the job is rejected. The constraint set (20) makes w_{ij} to 1 when a task of job i is assigned to resource j.

6.1. Solution methodology

CPLEX was unable to solve this model optimally even for problems with 10 jobs and 10 resources. So we develop an efficient HRED-T heuristic which is an extension of the HRED heuristic. In the HRED-T heuristic, we calculate the rank for each combination of the task and resource similar to the HRED heuristic. The main difference is that the HRED-T heuristic also includes the integration cost in the calculation of rank. Moreover, after assigning a task of one job, HRED-T assigns all other tasks of the same job before making assignments for any task of other jobs. The formal algorithm for HRED-T is given below.

Algorithm HRED-T

Step 1: Calculate the rank $V_{ijl} = (L_i - t_{ijl}p_j)$ for each combination of job i, task l, and resource j, and sort them in non-increasing order of V_{ijl}. Here, $i = 1, 2, \dots, m$; $j = 1, 2, \dots, n$; and $l = 1, 2, \dots, e_i$. Let us refer to this list as List 1.

Step 2: Remove all those combinations from List 1 where V_{ijl} is negative or $t_{ijl}p_j > B_i$.

Step 3: Let α' be the task, β' be the job and γ' be the resource in the first combination of the updated List 1. Create a new list, called List 2, having all the combinations for all the tasks of job β' from List 1 in the same order.

Step 4: Let η be the set of tasks that have already been assigned to γ'. Combine all the tasks in set η with the task α' of job β' and sort all of them in non-decreasing order of their deadline and ready time. Now try to assign these tasks to resource γ' in the sorted order. If all of these tasks can be assigned without violating the deadline constraint of any of the jobs or the availability constraint of resource γ', then assign the task α' of job β' to resource γ' and proceed to the next step; otherwise remove the combination of task α' and resource γ' from List 2 and go to *Step 7*.

Step 5: Compute the total cost (including the integration cost) of all assigned tasks of job β'. If this cost is less than the budget of job β', then remove all combinations of task α' from List 2 and skip to *Step 7*; otherwise proceed to the next step.

Step 6: Remove the assignments for all tasks of job β'. Remove the combination for task α' of job β' and resource γ' from List 1. Empty List 2 and go to *Step 11*.

Step 7: If the updated List 2 is empty, then go to *Step 8*; otherwise skip to *Step 9*.

Step 8: If all tasks of job β' has been assigned, then remove all the combinations for each task of job β' from List 1 and go to *Step 11*; otherwise go to *Step 6*.

Step 9: For each combination of task l in List 2, update $V'_{\beta' jl}$ as follows: $V'_{\beta' jl} = (L_{\beta'} - t_{\beta' jl}p_j - \xi_{\beta' jl})$, where $\xi_{\beta' jl}$ is the additional integration cost incurred by job β' if the task l is assigned to resource j, given the current assignments for the tasks of job β'.

Step 10: Sort the combinations in List 2 in non-increasing order of updated $V'_{\beta' jl}$. Let α' be the task and γ' be the resource in the first combination of sorted List 2. Go to *Step 4*.

Step 11: If the updated List 1 is empty, then terminate; otherwise go to *Step 3*.

6.2. Performance of the HRED-T heuristic

For the IP formulation, CPLEX was unable to provide any bound in a reasonable amount of time. Therefore, similar to Eq. (16), we determine the percentage gap of the HRED-T solution from the lower bound solution obtained by the LP relaxation. The test bed is again chosen to balance the goals of realism and feasibility. Due to additional complexity of the problem (over the model presented in Section 3), CPLEX was unable to solve even the LP relaxation problem for larger size problems. The number of jobs and the number of resources were chosen based on what size of LP relaxation was possible to solve in a reasonable amount of time.

We consider three values each for the number of jobs (5, 10, and 15) and the number of resources (10, 15, and 20). Hence, there are nine (3×3) problem classes. For each problem class, we choose three values each for the range of the number of tasks e_i for each job i (1–4, 3–6 and 5–8) and the range of the unit integration cost C (0, 5, and 10). Therefore, there are nine (3×3) problem instances for each problem class. The ranges for other parameters are as follows: t_{ijl}–(2–20), p_j–(2–5), A_j–(0–30), U_j–(80–100), B_i–(120–180), D_i–(80–100), and L_i–(100–150). The actual parameter values are drawn from these ranges randomly.

Each row in Table 6 shows the result for one problem class. For each problem class, the *maximum*, *average* and *minimum* percentage gaps are reported over nine problem instances in that class. The average percentage gap lies between 3% and 5.1%. Following the discussion in Section 4.5.1, the gap of HRED-T solution from the LP bound may be higher than its gap from the optimal solution because of the gap between the LP bound and the optimal solution. However, it is difficult to verify because we are unable to obtain the optimal solution even for small size problems. Since the minimum gap in each row is zero, the HRED-T heuristic provides the optimal solution for at least one problem instance in each problem class.

Table 6
Performance of the HRED-T heuristic

Problem class	Number of jobs	Number of resources	Percentage gap of HRED-T cost from LP bound		
			Maximum	Average	Minimum
1	5	10	11.41	3.33	0
2		15	9.07	4.10	0
3		20	13.78	3.73	0
4	10	10	10.75	4.21	0
5		15	12.28	3.89	0
6		20	12.73	4.93	0
7	15	10	10.96	4.65	0
8		15	15.84	5.09	0
9		20	16.01	5.02	0

6.3. Discussions and managerial implications

The two important parameters in this model are: (i) the number of tasks for a job, indicating the degree of parallelism, and (ii) the unit integration cost C. In this section, we present the sensitivity of the HRED-T solution to these two parameters. For this experiment, the parameter values are chosen randomly from the following ranges: $B_i \in$ [5000–8000], $D_i \in$ [50–100], $L_i \in$ [100–150], $t_{ijl} \in$ [15–20], $p_j \in$ [2–20], $R_i \in$ [0–10], $C\in$[20–80], and Number of tasks [10–30].

As the number of tasks increases, the job owners gains more benefit of parallelism. Therefore, we find that the cost decreases with the number of tasks (see Fig. 7a). In this figure, we also present the cost for the non-distributed processing where all the tasks of a job are assigned together to one resource (i.e., the model considered in Section 4). The non-distributed processing model always has higher cost than the distributed processing model, because it is not able to exploit the benefits of parallelism. From a mathematical perspective, the distributed model is a relaxation of the non-distributed model: a non-distributed solution is always feasible for the distributed problem. Hence, the objective function values obtained by the distributed model are guaranteed to be superior.

However, in the distributed processing model the job owners incur an additional integration cost. The impact of this cost is illustrated in Fig. 7b. The revenue and cost in this figure are normalized in order to plot them on the same scale as the number of rejections. Clearly, the cost increases with the integration cost. The number of rejections is constant initially, but it starts increasing as the integration cost increases beyond a threshold. When the integration cost increases, it becomes less beneficial for the job owners to assign their tasks to different resources. Hence, the increase in integration cost benefits the resource owners. Therefore, the revenue of resource owners increases with the integration cost until the number of rejections is constant. However, when the number of rejections starts increasing, less jobs are available for resources, and hence the resource owner revenue stars decreasing.

Due to parallelism, the assignment of tasks in the distributed processing model is much easier as compared to that in the non-distributed processing model. Hence, the number of rejections in the distributed processing model is usually much less than that in the non-distributed

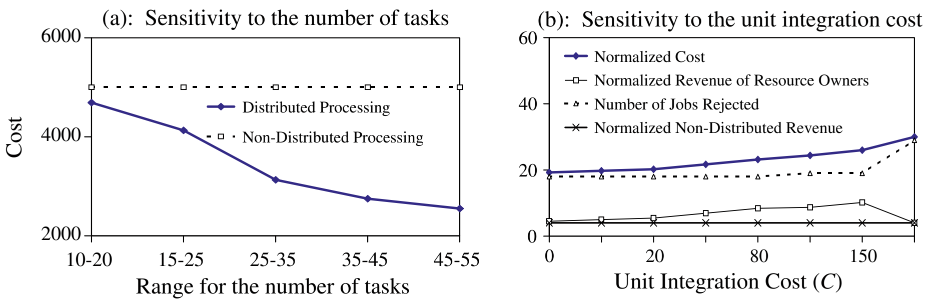

Fig. 7. Sensitivity to the parameters related to distributed processing. (a) Range for the number of tasks. (b) Unit integration cost (C).

processing model. Hence, Fig. 7b shows that the revenue of resource owners in distributed processing model is always higher than that in the non-distributed processing model. Moreover, the revenues are same in both models when the integration cost is high. In this case, the distributed processing model works as a non-distributed model by assigning all tasks of a job to the same resource.

7. Real-time scheduling in grid computing

In this section, we consider a real-time scheduling problem in grid computing. It is possible in a grid that the jobs arrive in real-time rather than in large batches. In such grids, it is very important to find the assignment efficiently and effectively. As discussed earlier, due to their computational efficiencies, HRED and HRED-T heuristics can be effectively used for the real-time scheduling.

Here, we compare the performance of two models widely used in the real-time scheduling of grid computing: (i) a non-preemptive model, and (ii) a preemptive model (Private Communications). In the non-preemptive model, the grid owner cannot reject a job once it is accepted. However, upon the arrival of a new job in the grid, the grid owner may sometimes find it beneficial to reject a previously accepted job and accept the new job. The preemptive model provides such flexibility to the grid owner. In these preemptive models, the grid owners usually incur an additional cost of preempting a job which depends upon the agreement between the job owner and the grid owner.

In Fig. 8, the cost is the sum of the cost of assigning jobs to resources and the cost of preempting jobs. In this experiment, the number of jobs and the number of resources are 100 and 30, respectively. The ranges for other parameter values are as follows: t_{ijl}–(2–20), p_j–(5–20), A_j–(0–30), U_j–(70–100), B_i–(100–200), D_i–(30–100), and L_i–(0–150). The actual parameter values are drawn from these ranges randomly. The preemptive cost of a job is drawn from the ranges shown in Fig. 8. For each range, we run the preemptive model five times and compute the average cost across those five instances, which is plotted in Fig. 8. We also present the cost of the non-preemptive model along with the average cost of the preemptive model in Fig. 8.

Since the preemptive model is a relaxed version of the non-preemptive model, the cost for the preemptive model

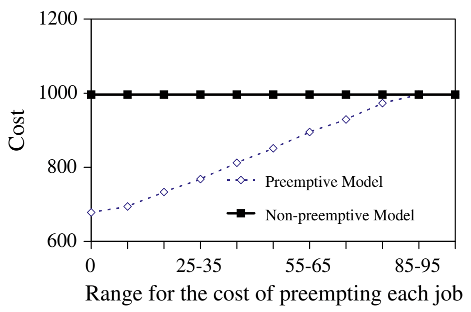

Fig. 8. Impact of the cost of preempting a job.

is always less than the cost for the non-preemptive model. However, as the cost of preempting a job increases, the preemptive model becomes less lucrative, and its cost becomes same as the cost of the non-preemptive model when the cost of preemptive a job is comparably very high.

Using this study, the grid owners can decide the cost of preempting a job which is beneficial for them as well as negotiable with the job owner. It would also help the grid owners in making decisions about which model to use. If they find that the benefit gained from using the preemptive model is not significant enough, then it may be better for them to use the non-preemptive model because the preemptive model involves additional complexity in managing the system.

8. Conclusions and future research directions

The success of grid computing in enterprises will depend on the cost effective usage of the system for various computationally intensive jobs. Given the vast number of resources that are available on an enterprise wide grid, an important problem is the scheduling of tasks on the grid with various business objectives. In this paper, we present a mathematical model for this problem and show that it is not only strongly NP-hard, but it is also non-approximable. Hence, we propose the efficient heuristics to find the near-optimal solution.

This paper provides managerial insights for all three entities involved in the grid computing – job owners, resource owners, and the scheduler. We consider two variants of the resource allocation problem in grid computing. First, we consider that all tasks of a job are assigned to the same resource. We show that this problem is strongly NP-hard and present an efficient heuristic called Highest Rank Earliest Deadline (HRED). The experiments show that the proposed HRED heuristic provides the near-optimal solution for a wide variety of problem instances. We also present several managerial insights by studying the impact of various parameters on the solution.

Second, we consider the problem where different tasks of a job can be assigned to different resources. For this problem, we propose another efficient heuristic called HRED-T, which is an extension of the HRED heuristic. This heuristic is also shown to provide very good results for the problems considered in our experiment. Based on the results of this heuristic, we again provide several managerial insights. Finally, we compare the performance of two widely used models in the real-time scheduling of grid computing and present the useful implications for managers.

Now, we discuss some of the limitations of this study and present some possible directions for future research. In this study, we assume that there is no precedence constraint among different jobs or different tasks of a job. Usually the jobs are independent of each other in the grid, but different tasks of a job may have some precedence constraints. Hence, it is an interesting direction for future research. Moreover, the jobs coming from the same division of an organization may have some interdependency in the penalty costs. Although it can be captured up to some extent by assigning the penalty costs appropriately, a more comprehensive model with explicit dependencies in terms of the penalty costs would be interesting to study. Such dependencies will not only make the problem extremely difficult to solve, but would also require estimating a very large number of parameters. Other interesting directions for future study are to design some constant bound polynomial approximation algorithms and compare the performances of different algorithms.

References

Angulo, D., Foster, I., Liu, C., Yang, L., 2002. Design and evaluation of a resource selection framework for grid applications. In: Proceedings of IEEE International Symposium on High Performance Distributed Computing.

Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M., 1992. Proof verification and hardness of approximation problems. In: Proceedings of the 33rd IEEE Symposium on the Foundations of Computer Science, pp. 14–23.

Bapna, R., Das, S., Garfinkel, R., Stallaert, J., 2008. A market design for grid computing. Informs Journal on Computing 20 (1).

Beltrame, G., Brandolese, C., Fornaciari, W., Salice, F., Sciuto, D., Trianni, V., 2001. Dynamic modeling of inter-instruction effects for execution time estimation. In: Proceedings of the 14th International Symposium on System Synthesis, pp. 136–141.

Berman, F., Fox, G., Hey, T., 2003. The grid: Past, present, future. Grid Computing – Making the Global Infrastructure a Reality. John Wiley & Sons, pp. 9–50.

BioGrid 2001. Virtual Lab Tools for Data Intensive Computing on Grid: Drug Design Case Study. Rajkumar Buyya. <http://www.buyya.com/talks/biogrid.ppt>.

Brandolese, C., Fornaciari, W., Salice, F., Sciuto, D. 2001. Source-level execution time estimation of C programs. In: Proceedings of the Ninth International Symposium on Hardware/Software Codesign, pp. 98–103.

Buyya, R., Abramson, D., Giddy, J., 2000. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In: Proceedings of 4th International Conference on High Performance Computing.

Buyya, R., Abramson, D., Giddy, J., Stockinger, H., 2002a. Economic models for resource management and scheduling in grid. Computing Concurrency and Computation Practice and Experience 14, 1507–1542.

Buyya, R., Murshed, M., Abramson, D., 2002b. A deadline and budget constrained cost-time optimization algorithm for scheduling task farming applications on global grids. Technical Report CSSE-2002/109, Monash University Melbourne Australia.

Buyya, R., Murshed, M., Abramson, D., Venugopal, S., 2005. Scheduling parameter sweep applications on global grids: A deadline and budget constrained costtime optimization algorithm. Software-Practice and Experiences 35 (5), 491–512.

Castro-Leon, E., Munter, J., 2005. Grid computing looking forward. Intel White Paper. <ftp://download.intel.com/business/bss/technologies/grid/grid_computing.pdf>.

Chekuri, C., Motwani, R., Natarajan, B., Stein, C., 2001. Approximation techniques for average completion time scheduling. SIAM Journal on Computing 31 (1), 146–166.

Cheliotis, G., Kenyon, C., Buyya, R., 2005. Ten lessons from finance for commercial sharing of IT resources. In: Subramanian, R., Goodman, B.D. (Eds.), Peer-to-peer Computing: The Evolution of a Disruptive Technology. Idea Group Inc. (IGI), pp. 244–264.

Coddington, P., 2002. DISCWorld, virtual data grids and grid applications. <http://www.gridbus.org/ozgrid/DiscWorld.ppt/>.

De Roure, D., Baker, M.A., Jennings, N.R., Shadbolt, N.R., 2003. The evolution of the grid. Grid Computing – Making the Global Infrastructure a Reality. John Wiley & Sons, pp. 65–100.

Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., Streit, A., 2002. On advantages of grid computing for parallel job scheduling. In: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 39–46.

Foster, I., Kesselman, C., Nick, J., Tuecke, S., 2002. The physiology of the grid: An open grid services architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum. <http://www.globus.org/alliance/publications/papers/ogsa.pdf>.

Garey, M.R., Johnson, D.S., 1978. Strong NP-completeness results: Motivation, examples, and implications. Journal of the ACM 25 (3), 499–508.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. WH Freeman and Company, New York.

Glasgow, B., 2003. Information technology insights: Grid computing moves into the next wave. Chemical Market Reporter 246, 17–22.

Globus Alliance 2007. University of Chicago. <http://www.globus.org/>.

Halang, W.A., 1989. A priori execution time analysis for parallel processes. In: Proceedings of Euromicro Workshop on Real Time, pp. 62–65.

Hall, L.A., Schulz, A.S., Shmoys, D.B., Wein, J., 1997. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. Mathematics of Operations Research 22 (3), 513–544.

Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour, R., 2000. Evaluation of job-scheduling strategies for grid computing. In: Proceedings of 1st IEEE/ACM International Workshop on Grid Computing, Lecture Notes in Computer Science 1971, pp. 191–202.

He, X., Sun, X., Laszewski, G.V., 2003. QoS guided min–min heuristic for grid task scheduling. Journal of Computer Science and Technology 18, 442–451.

Herroelen, W., De Reyck, B., Demeulemeester, E., 1998. Resource-constrained project scheduling: A survey of recent developments. Computers and Operations Research 25, 279–302.

Hoogeveen, H., Schuurman, P., Woeginger, G.J., 2001. Non-approximability results for scheduling problems with minsum criteria. Informs Journal on Computing 13 (2), 157–168.

Intel. 2004a. It's a grid world: The convergence of computing and communications. Intel White Paper. <http://www.intel.com/business/bss/technologies/grid/grid_computing.pdf>.

Intel. 2004b. Grid computing and finance. Presented at 2004 High Performance Technology on Wall Street, New York City. <ftp://download.intel.com/business/bss/technologies/grid/hpc_whitepaper.pdf>.

Krishnaswamy, S., Loke, S.W., Zaslavsky, A., 2004. Estimating computation times of data-intensive applications. IEEE Distributed System Online 5.

Lee, C., Lei, L., Pinedo, M., 1997. Current trends in deterministic scheduling. Annals of Operations Research 70, 1–41.

Legion. 2007. A world wide virtual computer. University of Virginia. <http://legion.virginia.edu/>.

Li, K., 2005. Job scheduling for grid computing on metacomputers. Journal of Parallel and Distributed Computing 65 (11), 1406–1418.

Min, R., Maheswaran, M., 2002. Scheduling co-reservations with priorities in grid computing system. In: Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02).

Mokotoff, E., 2001. Parallel machine scheduling problems: A survey. Asia-Pacific Journal of Operational Research 18, 193–242.

Papadimitriou, C.H., Yannakakis, M., 1991. Optimization, approximation, and complexity classes. Journal of Computer and System Sciences 43, 425–440.

Pinedo, M., 1995. Scheduling: Theory, Algorithms, and Systems. Prentice Hall, New Jersey.

Schnizler, B., Neumann, D., Veit, D., Weinhardt, C., 2008. Trading grid services – A multi-attribute combinatorial approach. European Journal of Operational Research 187 (3), 943–961.

Seymour, K., YarKhan, A., Agrawal, S., Dongarra, J., 2005. NetSolve: Grid enabling scientific computing environments. In: Grandinetti, L. (Ed.), Grid Computing and New Frontiers of High Performance Processing. Elsevier.

SPEC 2007. Standard Performance Evaluation Corporation. <http://www.spec.org/>.

Sun Grid 2006. Sun grid computing utility. Sun Microsystems. <http://www.network.com>.

Tsunamic Technologies, Inc. (TTI) 2006. Cluster computing on demand. <http://www.tsunamictechnologies.com/services/services.htm>.

Utility Computing 2006. Utility computing. Sun Microsystems. <http://www.sun.com/service/sungrid/overview.jsp>.

Wolski, R., Brevik, J., Plank, J.S., Bryan, T., 2003. Grid resource allocation and control using computational economies. Grid Computing – Making the Global Infrastructure a Reality. John Wiley & Sons, pp. 747–771.

World Wide Grid 2007. Global data-intensive grid collaboration. Available at <http://gridbus.cs.mu.oz.au/sc2003/list.html>.

Yang, J., Khokhar, A., Sheikh, S., Ghafoor, A. 1994. Estimating execution time for parallel tasks in heterogeneous processing (HP) environment. In: Proceedings of Heterogeneous Computing Workshop, pp. 23–28.

Private communications

Professor Martin Bichler, Roland Berger & O_2 Germany Chair, Department of Informatics, Technische Universitat Munchen, Germany, bichler@in.tum.de.

Julio Ibarra, Executive Director, NSF AMPATH Project, Julio.Ibarra@fiu.edu.

Anthony Skipper, Vice President, Merrill Lynch, IDS AIS PE, anthony_skipper@ml.com.

Art Vandenberg, Director, Advanced Computer Services, Georgia State University, avandenberg@gsu.edu.