Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 23 (2007) 163–178
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Efficient task replication and management for adaptive fault tolerance in Mobile Grid environments

Antonios Litke*, Dimitrios Skoutas, Konstantinos Tserpes, Theodora Varvarigou

Department of Electrical and Computer Engineering, National Technical University of Athens, 9, Heroon Polytechniou Str, 15773 Athens, Greece

Received 4 November 2005; received in revised form 17 April 2006; accepted 18 April 2006
Available online 19 June 2006

Abstract

Fault tolerant Grid computing is of vital importance as the Grid and Mobile computing worlds converge to the Mobile Grid computing paradigm. We present an efficient scheme based on task replication, which utilizes the Weibull reliability function for the Grid resources so as to estimate the number of replicas that are going to be scheduled in order to guarantee a specific fault tolerance level for the Grid environment. The additional workload that is produced by the replication is handled by a resource management scheme which is based on the knapsack formulation and which aims to maximize the utilization and profit of the Grid infrastructure. The proposed model has been evaluated through simulation and has shown its efficiency for being used in a middleware approach in future mobile Grid environments.

Keywords: Fault tolerance; Scheduling; Grid computing

1. Introduction

Grid computing has recently migrated from traditional high-performance and distributed computing to pervasive and utility computing based on the advanced capabilities of wireless networks and lightweight, thin devices. As a result, this has seen the emergence of a new computing paradigm, which is the Mobile Grid. Mobile Grid is a full inheritor of the Grid, with the additional feature of supporting mobile users and resources in a seamless, transparent, secure and efficient way. It has the ability to deploy underlying ad hoc networks and provide a self-configuring Grid system of mobile resources (hosts and users) connected by wireless links and forming arbitrary and unpredictable topologies. However, it is also the basis and the enabling technology for pervasive and utility computing due to the ability of being open, highly heterogeneous and scalable.

This modern approach, which combines thousands of parts into a large system, provides generally less reliable platforms which, in combination with the long running codes, results in application execution times that exceed the mean time to failure of the machines. For this reason, *fault tolerance* is of vital importance in this new Mobile Grid paradigm, since both mission-critical systems (such as air traffic control systems, where no down time can take place) and computationally intensive applications (where the lost work from the failure has to be minimized) belong in the context of diverse, dependable and cross-organizational environments, which is the case for the emerging Mobile Grid. Mobile Grid computing is not expected to be fault free, despite the fact that individual techniques such as *fault avoidance* and *fault removal* [1] may additionally be applied to its resources. Therefore, fault tolerance mechanisms need to be deployed to allow the Grid system to perform correctly in the presence of faults, enhancing it with the appropriate reliability.

In this paper we present a fault tolerant model for task scheduling in Mobile Grid systems based on the task replication concept. The Mobile Grid is a dynamic system where environmental conditions are subject to unpredictable changes: system or network failures, system performance degradation, removal of machines, variations in the cost of resources, etc. In such a context, task replication and scheduling is an efficient way to guarantee the completion of tasks according to the restrictions set by the users. The basic idea is to produce

* Corresponding author. Tel.: +30 210 7722558.
E-mail addresses: ali@telecom.ntua.gr (A. Litke), dskoutas@telecom.ntua.gr (D. Skoutas), tserpes@telecom.ntua.gr (K. Tserpes), dora@telecom.ntua.gr (T. Varvarigou).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.04.014

and schedule in the Grid infrastructure multiple replicas of a given task. The number of replicas is calculated by the Grid middleware and is based on the failure probability of the Grid resources and the policy that is adopted for providing a specific level of fault tolerance. The adopted replication model is based on *static replication* [2], meaning that, when a replica fails, it is not substituted by a new one. The introduction of task replicas causes an overhead in the *workload* that is allocated to the Grid for execution. Scheduling and resource management are important in optimizing Grid resource allocation [3] and determining its ability to deliver the negotiated Quality of Service to all users [4,5]. The idea in this paper is to handle the additional tasks with a resource management scheme based on the knapsack problem formulation, where to each task a *weight* and a *profit* for its correct execution have been assigned. By this, we allow for an efficient time scheduling of the tasks and their replicas so as to maximize the utilization of the Grid resources and the profit that we can gain from the successfully executed ones. For this formulation, we use four different algorithms (*Backtracking, Branch and Bound, Dynamic Programming*, and a heuristic *Greedy* algorithm) so as to study their behavior in various instances and to identify their performance in this difficult problem [6].

The remainder of the paper is structured as follows. Section 2 presents related work in the field of fault tolerance in Grid computing as well as in Grid scheduling and formulation such as the knapsack problem. Section 3 gives an overview of the problem formulation and the notation that is followed in the paper. Section 4 provides the concept and analysis of the task replication model adopted for achieving the desired fault tolerance in the Grid infrastructure. Section 5 presents the scheduling and resource management model, which includes the functionality of handling the replicas of each task in an efficient way. In Section 6, we present the simulation results in terms of time efficiency of the various algorithms adopted, as well as the overall evaluation of the proposed scheme in terms of performance for profit optimization. Finally, Section 7 concludes with a discussion on future research and potential for the current study.

2. Related work

There are various approaches to make Grid computing fault tolerant [1,7,5]. The basics, however, are *checkpoint recovery* and *task replication*. The former is a common method for ensuring the progress of a long-running application by taking a checkpoint, i.e. saving its state on permanent storage periodically. A checkpoint recovery is an insurance policy against failures. In the event of a failure, the application can be rolled back and restarted from its last checkpoint—thereby bounding the amount of lost work to be re-computed. Task replication is another common method that aims to provide fault tolerance in distributed environments by scheduling redundant copies of the tasks, so as to increase the probability of having at least a simple task executed. A brief overview of the options in fault tolerant computing on the Grid can be found in [7]. Moreover, in [8] a very interesting analysis of requirements for fault tolerance in the Grid is presented, along with a failure detection service and a flexible failure handling framework.

There has been a variety of implementations that have addressed the problem of fault tolerance in Grid and distributed systems. Globus [9] provides a heartbeat service to monitor running processes to detect faults. The application is notified of the failure and is expected to take appropriate recovery action. Legion [10,11] provides mechanisms to support fault tolerance such as checkpointing. Other Grid systems like Netsolve [12,13], Mentat [14] and Condor-G [15] have their failure detection and failure recovery mechanisms. They provide a single user-transparent failure recovery mechanism (e.g. re-trying in Netsolve and in Condor-G; replication in Mentat). FATCOP [16] is a parallel mixed integer program solver that works in an opportunistic computing environment provided by the Condor resource management system, using an implementation of a branch-and-bound algorithm. Peer-to-peer (P2P) systems also follow various methods for fault tolerance in their operation. An interesting overview of P2P systems is presented in [17] and [18].

The difference between these systems and our proposed scheme lies in the fact that the one presented here addresses fault tolerance as a metric that is adjusted in the Grid infrastructure based on the reliability model of its resources. Moreover, it applies to all Grid environments and is especially beneficial for low-workload jobs in unreliable environments, such as Mobile Grids [19,20], which consist of mobile resources (hosts and users) connected by wireless links and forming arbitrary and unpredictable topologies.

As far as it concerns scheduling in Grid, there are several systems that have been developed. The most significant attempts can be found in meta-schedulers such as Nimrod-G [21,22], software execution environments such as GRaDS [23] and task brokers such as Condor-G [24]. The latter is a product of a much more complicated entity that consolidates scheduling policies which comprised specialized workload management systems. Additionally, AppLeS [25] is a scheduling system which primarily focuses on developing scheduling agents for individual applications on production. Other interesting works on scheduling and meta-scheduling are presented in [26] and [27] where, in the former, the authors present a heuristic scheduling of bag-of-tasks with QoS constraints, while the latter handles the problem of distributed job scheduling in Grids using multiple simultaneous requests. However, in coherent, integrated Grid environments (such as Globus [9] and Unicore [28]) there are also scheduling and resource management techniques applicable in a more standard manner.

Finally, other studies have also addressed resource management in Grids, such as the knapsack formulation problem. In [29], the resource allocation in a Grid environment is formulated as a knapsack problem and techniques are developed and deployed so as to maintain the QoS properties of a schedule and, at the same time, to maximize the utilization of the grid resources. This approach, although presenting very interesting results, does not take into account the mobility aspect of the Grid (having unreliable links) and, moreover, the

reliability model of the Grid infrastructure so as to provide a fault tolerance threshold for the tasks that are going to be executed.

3. Notation and problem formulation

We consider a Grid infrastructure consisting of a set of N_P computational machines that are either fixed or mobile (like personal digital assistants, personal computers, notebooks, workstations) which will also be mentioned as *resources* of the mobile Grid. Each resource has a *computational rate* which is denoted as μ_j, $j \in \{1, 2, \ldots, N_P\}$, and represents the ability of this resource to perform specific operations in a given time period (for instance, million floating operations per second—MFLOPS). The aggregate computational rate of the Grid is $\mu_G = \sum_{j=1}^{N_P} \mu_j$.

We also consider a set of N different tasks, $T_i, i \in \{1, 2, \ldots, N\}$, which are to be assigned to the Grid for execution. We assume that the tasks are non-preemptable and non-interruptible [30,31]. This means that a task cannot be broken into smaller sub-tasks or modules and it has to be executed as a whole on a single processor of a given resource. Additionally, as soon as a task starts its execution on a processor, it cannot be interrupted and it occupies the whole processor until its execution completes successfully or a failure occurs. In the analysis that we follow in this paper, we consider typical Grid tasks as they are met in various Grid platforms (such as Globus). These tasks can be executables together with their input files and parameters, or only executables submitted for remote execution, or only input files that are going to be processed on a remote machine. Each task T_i has an *execution time* ET_i and a *deadline* D_i. The execution time ET_i corresponds to the time interval that the execution of T_i lasts if it is executed in a processor of unitary rate $\mu = 1$. Otherwise, we denote as ET_{ij} the time interval that the task T_i lasts on resource j with computational rate μ_j. The deadline D_i of the task T_i represents the latest time at which the Grid has to deliver the results to the user. It is a quantity specified by the end-user, who is willing to pay for the resources that it has consumed.

We introduce the concept of *workload*, $w_i, i \in \{1, 2, \ldots, N\}$, which is the load of computational work that task T_i stresses on a resource for its successful execution. We have:

$$w_i = \mu_j ET_{ij} \tag{1}$$

which is measured in *computational units* (such as million floating point operations—MFLO). In the special case of having a resource with a unitary rate, we get $w_i = ET_i$, which represents the case that the workload of the task T_i when executed on unitary rate resource is equal to the execution time.

Finally, each task T_i is associated with a *profit* v_i, indicating the revenue produced by the successful execution of the task. This profit is measured in *currency units* and may refer to the price that the user is paying for the service provided. It may also be a more abstract notion of value, incorporating revenue in other situations (for example, in a crisis management scenario, or in a workflow of orchestrated services, a task which produces results that help to handle the crisis or is critical for the sequel of the workflow may not be charged — similarly to as in emergency calls — but still its execution yields a very high value due to the importance of the situation which can be translated in a virtual profit).

During task execution on the mobile Grid, various errors might occur. These kinds of errors are commonly met in distributed systems as well as in Grid environments [2,7,32,33], and most often include: (i) *Crash failures*, where a server halts, but was working correctly until that moment; (ii) *Omission failures*, where a server fails to respond to incoming requests and to send messages; (iii) *Timing failures*, where the server succeeds in responding but its response is beyond the specified time interval.

We define as *failure probability*, Pf_i, of a task T_i the probability that the task fails to be executed on the Grid. The proposed *fault tolerance* scheme in this paper is based on the notion of replication. According to that, task *replicas* are generated by the Grid middleware and are assigned to the resources for execution. The term *replica* or *task replica* is used to denote an identical copy of the original task. By producing task replicas, the failure probability for each task can be lowered significantly; however, the number of tasks that are finally assigned to the Mobile Grid increases, increasing respectively the total workload that is assigned to the Grid for execution. We assume that m_i replicas — denoted by $T_{ik}, k = 1, \ldots, m_i$ — of a task T_i are produced and are placed among other tasks and replicas that are going to be assigned to a processor of a mobile node by the scheduler. The number of tasks, including their replicas, will be denoted N'. Therefore, $N' = N + \sum_{i=1}^{N} m_i$.

Given the failure probability Pf_{ik} of each one of the m_i replicas T_{ik} of task T_i, the new failure probability Pf_i' for task T_i is:

$$Pf_i' = Pf_i \cdot \prod_{k=1}^{m_i} Pf_{ik}. \tag{2}$$

The above corresponds to the probability of the event "all the replicas and the original task fail". Respectively, the *success probability* is equal to the probability of the event "the original task or at least one of its replicas executes successfully". The number of replicas issued depends on the failure probabilities of the original tasks and on the desired fault tolerance level in the Grid infrastructure. We denote λ as the desired failure probability threshold. Thus, for each task, a sufficient number of replicas is created to satisfy the condition:

$$Pf_i' \leq \lambda \tag{3}$$

where $\lambda \in (0, 1)$.

In this paper, we examine the failure probability of a task with respect to the mobile resource it is assigned for execution. Based on the various types of failures, we assume that a task or a replica fails if the resource on which it is assigned fails during the execution time. This failure is not necessarily a processor (crash) failure, but can also be an omission (such as time omission) due to an unreliable connection. For this reason, the *mean time to failure* (MTTF) is used, which is the expected time

that a mobile Grid resource will function before it fails for any reason. Conventionally, MTTF refers to non-repairable objects, while *mean time between failure* (MTBF) refers to repairable objects. In large and complex systems such as Mobile Grids, although the individual component reliabilities can be high, the overall reliability of the system can possibly be low [34], due to coupled failure modes or various extrinsic factors. The lifetime of the Mobile Grid resources under discussion is assumed to start at time $t = 0$, and any events occurring at times $t < 0$ are irrelevant.

4. Task replication model

The model that is used is based on the *Weibull distribution*, since this model effectively represents the machine availability in large-scale computing environments [35] such as wide-area enterprises, the Internet and Grids. The Weibull distribution is often used to model the time until failure of many different physical systems and, for this reason, it is more suitable for our study compared to other distributions such as the Poisson or exponential distributions. The parameters in the distribution provide a great deal of flexibility to model systems in which the number of failures increases over time (e.g. bearing wear), decreases over time (e.g. some semiconductors), or remains constant with time (e.g. failures caused by reasons that are external to the system) [36].

4.1. The Weibull distribution

If X is a random variable, then the *probability density function* (*pdf*)

$$f(x) = \frac{\beta}{\delta}\left(\frac{x}{\delta}\right)^{\beta-1} \mathrm{e}^{-(\frac{x}{\delta})^{\beta}} \tag{4}$$

is a Weibull distribution for $x > 0$, with *scale* parameter $\delta > 0$ and *shape* parameter $\beta > 0$. In particular, for the case $\beta = 1$, the Weibull distribution is identical to the *exponential distribution*.

The *cumulative distribution function* (*cdf*) of X is:

$$F(x) = 1 - \mathrm{e}^{-(\frac{x}{\delta})^{\beta}}. \tag{5}$$

The reliability function $R(x)$ of a machine that follows a specific distribution is the probability that the machine does not fail in the time interval $(0, x]$. It is one minus the *cdf*, and thus, for the Weibull distribution, is given by (see Fig. 1):

$$R(x) = \mathrm{e}^{-(\frac{x}{\delta})^{\beta}}. \tag{6}$$

A system that follows a Weibull failure law (WFL) has a *hazard function* (instantaneous failure rate function) [37,38] of the form:

$$h(x) = \rho\alpha x^{\alpha-1} \tag{7}$$

where α and $\rho > 0$. There are three cases, according to the values of α.

- If $\alpha = 1$, then the hazard rate is constant and the Weibull failure law is an Exponential Failure Law.

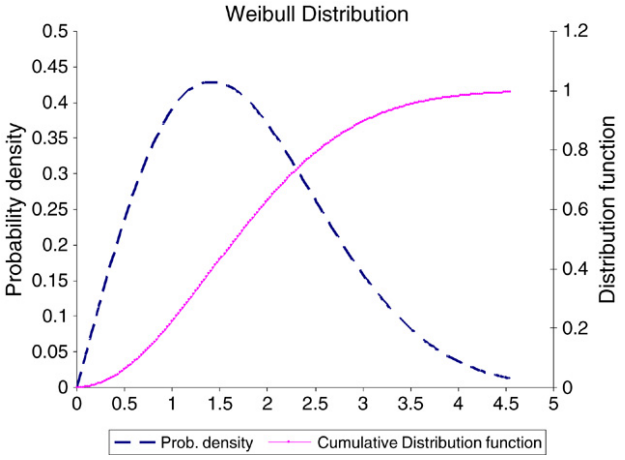


Fig. 1. The Weibull distribution. The probability density function and cumulative distribution function for $\beta = \delta = 2$.

- If $\alpha > 1$, the hazard function increases as the system gets older. In most areas of application, this is the most common case. In this case, the system has memory and remembers its age.
- If $\alpha < 1$, the hazard function decreases as the system ages.

4.2. Estimation of the Weibull parameters

The estimation of the Weibull parameters is of vital importance for the formation of the reliability function and thus for the replication procedure itself. There are several approaches for the estimation of the Weibull parameters, such as *Probability Plotting, Maximum Likelihood Estimation (MLE)* and *Least Square Errors (LSE)* [39]. Since the goal is to have an automated way to calculate the parameters, the probability plotting method is omitted. The most convenient method for the calculation of the parameters is linear regression, which allows for a straightforward calculation of the Weibull parameters. We assume that the MTBF is distributed as the two-parameter Weibull *pdf*. The actual times before failure are collected in ascending order. We convert (5) into a linear equation of the form $y = \beta x + b$. In particular, and for convenience, we take:

$$\ln\left(\ln\left(\frac{1}{1 - F(t)}\right)\right) = \beta \ln(t) - \beta \ln(\delta). \tag{8}$$

The data that we obtained should closely follow this line. We obtain the coordinates for the data points respectively:

$$x_i = \ln(t_i), \quad y_i = \ln\left(\ln\left(\frac{1}{1 - mr_i}\right)\right) \tag{9}$$

where mr_i stands for the median rank of the data point (x_i, y_i). Median ranks estimate the unreliability for each failure and can generally be described as the true probability that should be for i failures out of n samples at 50% confidence. A quick method for computing the median rank is:

$$mr_i = \frac{i - 0.3}{n + 0.4}. \tag{10}$$

By simple linear regression, we can perform a least-squares fit on the sequence of the data points collected. Using the equations for x_i and y_i, we get:

$$\beta = \frac{k\sum_{j=1}^{k} x_j y_j - \left(\sum_{j=1}^{k} x_j\right)\cdot\left(\sum_{j=1}^{k} y_j\right)}{k\sum_{j=1}^{k} x_j^2 - \left(\sum_{j=1}^{k} x_j\right)^2},$$
$$\delta = \exp\left(\frac{\left(\sum_{j=1}^{k} y_j\right)\cdot\left(\sum_{j=1}^{k} x_j^2\right) - \left(\sum_{j=1}^{k} x_j\right)\cdot\left(\sum_{j=1}^{k} x_j y_j\right)}{-\beta\cdot\left(k\sum_{j=1}^{k} x_j^2 - \left(\sum_{j=1}^{k} y_j\right)^2\right)}\right) \quad (11)$$

where k is the number of points (i.e. number of failures) that have been used as input to the estimation of the Weibull parameters.

To illustrate the estimation with an example, we consider $n = 100$ machines (preferably notebooks) of a Grid infrastructure, which begin operational life at the moment $t = 0$. We suppose that the first $j = 4$ failures occur at the times t_j, where $t_1 = 47$ h, $t_2 = 122$ h, $t_3 = 187$ h, and $t_4 = 233$ h. This gives us four data points, which are presented in ascending order in the following table together with x_i and y_i, which have been derived through (9) and (10).

j	t_j	x_i	y_i
1	47	3.850148	−4.96234
2	122	4.804021	−4.07001
3	187	5.231109	−3.60231
4	233	5.451038	−3.28211

By applying linear regression, we can perform a fit of this sequence of $k = 4$ data points to a line. By using the equations of (11) for the parameters of the Weibull distribution, we get $\beta = 0.016$ and $\delta = 548.484$.

4.3. Calculation of the replicas

The probability of a task failing is equal to the probability of all its replicas failing, as has already been described in Section 3. In this paper, we deal with failures that occur due to the fact that we have unreliable machines in the mobile Grid infrastructure. We will omit other cases of failure caused by the network connections (such as timing failures) or failures from the application software, in order to simplify the study. However, the proposed model can be enhanced so as to incorporate failure during the submission of the task and the results in the execution time interval, and allowing an additional probability of having the failure from errors that occurred during execution due to the application software.

Irrelevant to the scheduling algorithm (our scheme is not limited by the scheduling algorithm that is adopted), we assume that a task is to be scheduled on the mobile Grid. In this case, we assume that a task (before the replication) is to be scheduled on the mobile Grid for execution. Each resource of the mobile Grid is characterized by a reliability function and thus a hazard function. In the general case and regardless of any scheduling and resource assignment policy, we assume that the task is going to start its execution at the moment t. We suppose that a mobile resource has survived until time t. The hazard function, $h(t)$, is its failure rate during the short time interval $(t, t + \Delta t]$. Given that a mobile resource is still alive at time t, the probability of failure during the next Δt time units can be expressed generally as follows:

$$P(t < s < t + \Delta t | s > t) = P(t < s < t + \Delta t)/P(s > t) \quad (12)$$

where s represents the time of failure. For small Δt, we get $h(t) = f(t)/R(t)$.

In the case of having the execution time of a task on a given resource, we can calculate this probability by having the following equation:

$$P(t_0 < s < t_0 + ET_{ij} | s > t_0) = P(t_0 < s < t_0 + ET_{ij})/P(s > t_0) \quad (13)$$

where t_0 is the time when the task T_i is assigned to the resource for execution and ET_{ij} is the execution time that it lasts on the specific resources.

Sometimes, and in order to estimate this probability, it is necessary to know a priori the execution time of a task on a given resource, or at least a good estimation. There are several ways to predict the execution time for a task/resource pair, which are presented in [40–45]. Some of them use the previous resource performance. For large applications or non-stable running environments, application-level prediction models are applied. Additionally, a nonlinear prediction model for legacy applications with unknown source code can be used so as to allow confidentiality for commercial applications [46].

In the case of having small execution times, we can use the hazard rate function for the WFL, or otherwise, in the case of having longer execution time and in order to be more accurate, we get:

$$P(t_0 < s < t_0 + ET_{ij} | s > t_0) = [F(t_0 + ET_{ij}) - F(t_0)]/R(s) = g(ET_{ij}, s). \quad (14)$$

In this case, the $g(\cdot, \cdot)$ function gives the conditional probability for having failure of the resource while executing the specific task. The calculation of the number of replicas is based on the condition:

$$\prod_{k=1}^{m_i} g_{ik} \leq \lambda \quad (15)$$

where g_{ik} represents the probability of $g(\cdot, \cdot)$ for the kth replica of task T_i, denoted as T_{ik}.

In the specific case where we have the same constant hazard rate for each resource, then the WFL becomes an Exponential Failure Law, which is a case that is identical to the normal operation of a machine (excluding the "burn in" and the "wear out" phases). In this case, for every replica T_{ik} we have $g_{ik} = g_i$

and we get:

$$g_i^{(m_i+1)} \leq \lambda \Rightarrow (m_i + 1) \cdot \log(g_i) \leq \log(\lambda) \tag{16}$$

and, given that $g_i < 1 \Rightarrow \log(g_i) < 0$, we have

$$m_i \geq \frac{\log(\lambda)}{\log(g_i)} - 1. \tag{17}$$

Finally, we conclude that the minimum number of replicas required to achieve a failure probability less than or equal to the given threshold λ is:

$$m_i = \left\lceil \frac{\log(\lambda)}{\log(g_i)} - 1 \right\rceil. \tag{18}$$

5. Scheduling and resource management model

As described in Section 3, the Grid is considered to have a total computational rate $\mu_G = \sum_{j=1}^{N_P} \mu_j$, where $\mu_j, j \in \{1, 2, \ldots, N_P\}$ is the computational rate of each one of the N_P contributing processors of the mobile nodes. Thus, in a period of time T_{per}, the capacity of the Grid, meaning the amount of computational work that the Grid can execute, is:

$$C = \mu_G \cdot T_{per} \tag{19}$$

measured in the same *computational units* as the workload (e.g. MFLO).

Each task T_i submitted for execution is characterized by a workload w_i, a failure probability Pf_i based on the failure probability of the resource it is assigned for execution, and a profit v_i that is achieved if its execution is completed successfully. The total workload for the system is $W = \sum_{i=1}^{N} w_i$.

In order to achieve a high Grid throughput which is translated to the maximum utilization of the Grid resources and to gain the maximum profit out of it, it is necessary to decide which tasks should be executed in the given time period T_{per} so as to fulfill these requirements. We suppose that the period is selected so as to fulfill the deadline constraint for the tasks set by the users. In the case of having a Grid capacity greater than the total workload of the tasks to be submitted ($W \leq C$), then all tasks can be submitted to the Grid resources for execution. However, if $W > C$, only a subset of the tasks may be selected. To optimize the profit from the task execution, a resource management mechanism is needed that will decide which tasks will be selected for execution. The objective of this mechanism is to select a subset of tasks $T' \subseteq T$, so that the total profit is maximized, given that $W'_T \leq C$.

This problem can be formulated as an instance of the Knapsack Problem, and in particular the 0/1 Knapsack Problem [47], which is defined as follows:

"There are $N \geq 0$ items and a knapsack of capacity $C > 0$. Each item $i \in [1, N]$ has a profit $v_i > 0$ and a weight $w_i > 0$. Find the selection of items ($z_i = 1$ if selected; $z_i = 0$ if not) so that $\sum_{i=1}^{N} z_i \cdot w_i \leq C$ and $\sum_{i=1}^{N} z_i \cdot v_i$ is maximized".

Our process becomes more complicated when replication mechanisms are employed. In this case, the total workload, denoted as W_R, for the system is the sum of the workload of each pending task, including their created replicas, thus $W_R = \sum_{i=1}^{N}(1 + m_i) \cdot w_i$, where m_i denotes the number of replicas created for task T_i. We will assume that the successful execution of more than one replica of the same task does not generate additional profit. This assumption has been made since, otherwise, a concrete economic driven analysis for the identification of an exact price would be needed, which is outside the scope in this paper. The total profit that can be gained by successfully executing the set of the N tasks is $V = \sum_{i=1}^{N} v_i$. However, the profit of the tasks inside the knapsack is $V_{knap} = \sum_{i=1}^{N} z_i \cdot v_i$.

In the case that no replicas are created, the expected profit $V_{Expected}$ is:

$$V_{Expected} = \sum_{i=1}^{N} z_i \cdot v_i \cdot (1 - Pf_i) \tag{20}$$

while, in the case of replication, the expected profit is:

$$V_{Expected} = \sum_{i=1}^{N} z_i \cdot v_i \cdot \left(1 - Pf_i \cdot \prod_{k=1}^{m_i} Pf_{ik}\right). \tag{21}$$

However, the *actual profit* for both cases can be formulated as:

$$V_{actual} = \sum_{i=1}^{N} z_i \cdot v_i \cdot \zeta_i \tag{22}$$

where $\zeta_i = 1$ if at least one of the copies (original or replica) of task T_i has executed successfully, otherwise $\zeta_i = 0$.

According to the pricing and charging policy adopted by the Grid system, the profit v_i of a task may be greater in the case of replication, either by a fixed amount or by an amount that depends on the number of replicas created, since an additional charge can be claimed for providing fault tolerance, which is regarded as a QoS aspect in the provision of services. In simple words, task replication increases the probability of some tasks to be left out, resulting in profit loss, while on the other hand it increases the probability of the selected tasks of executing successfully, resulting in profit gain. The decision on how to resolve this trade-off is made according to the values of the parameters involved (failure probability, workload compared to grid capacity, task profits) in each instance of the problem.

5.1. Description of the scheduling and resource management model

The idea behind the specific fault tolerance and resource management model is to enable various scheduling mechanisms that already exist in various Grid middleware implementations, so as to enhance the overall functionality with the fault tolerance attribute through the replication and assignment of redundant task copies to different Grid resources. In simple mobile Grid environments (such as the one depicted in Fig. 2), various failures can happen, resulting in deviation from the user deadlines but also the total omission of executing the task set by the mobile user. The task replication and resource management

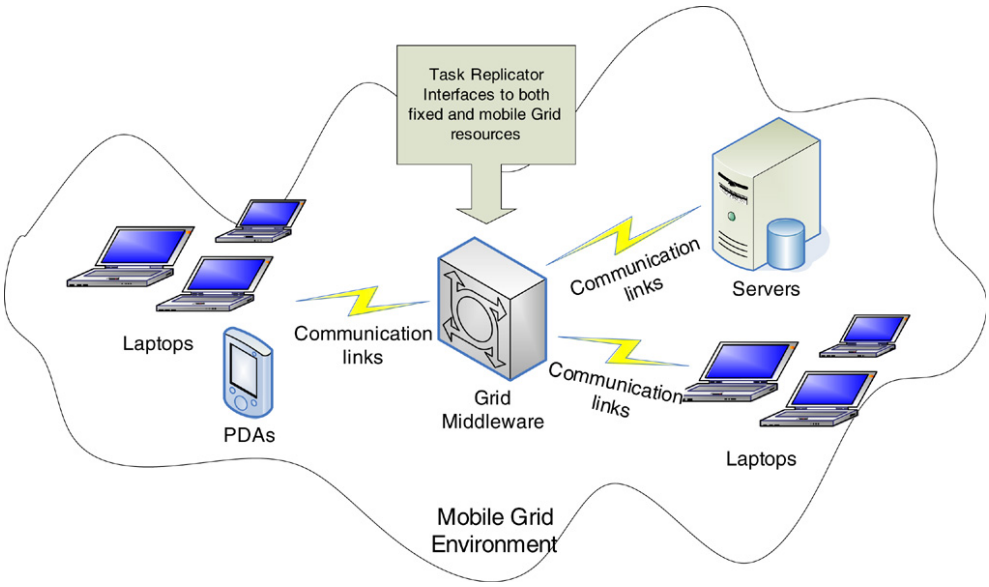


Fig. 2. A simple case of a Mobile Grid environment, indicating the framework of the task replication fault tolerance.

scheme based on the knapsack formulation that is presented in this paper is residing in the Grid middleware and is responsible for replicating tasks based on the reliability function of the resources with which it is associated.

As soon as a new resource is attached to the mobile Grid infrastructure (notebooks, but also fixed servers), the middleware is updated so as to estimate its Weibull parameters, based on the input that it will receive in the first failures (e.g. four failures). Having known the Weibull failure model for the given resource, it is able to estimate the failure probability for the next assignment of tasks for execution, and based on each task's estimated execution time on the specific resource.

The proposed fault tolerance and resource management scheme can collaborate with various scheduling mechanisms that would (based on their own criteria) produce a time schedule and processor assignment for the incoming tasks. Such mechanisms are derived by a much more complicated entity that consolidates scheduling policies, and examples of them can be found in various scheduling systems and Grid middleware solutions such as [9,21,23–25,28] and others. For each task that is assigned to a resource for execution, the proposed mechanism performs a check to identify the failure probability of the resource during the task execution. If inequality (3) is not met, then a new replica is produced and scheduled for execution on a new resource based on the scheduling scheme. The new failure probability of the replicated tasks based on Eq. (2) is then calculated, and this process is repeated until the total failure probability of all replicas of the task is less than the specified threshold λ.

As has been already mentioned in Section 3, there are various mechanisms for estimating the execution time of a given task on a given resource. Especially in commercial oriented Grid infrastructures, such mechanisms are mandatory for reasons related to the financial efficiency of the infrastructures, the execution planning of the incoming tasks, and the advanced reservation of resources for meeting the QoS constraints set by the users. However, the proposed scheme can have twofold functionality. It can schedule all the replicas for execution on the Grid resources regardless of the fact that, if the first succeeds, then the others can be removed from the queues waiting to be allocated. This approach implies a management overhead for the replicas, and an advanced monitoring system that will track the status of the execution procedure (resulting in a more complex solution to the mobile Grid environment) but saves computational resources (resulting in an increased efficiency). On the other hand, the fact that the replicas can be scheduled without any additional management for removing the multiple copies, while the first has completed successfully, is simple in terms of administration but makes a greedier consumption of the available resources in order to meet the desired fault tolerance.

The tasks that have been left out of the knapsack will be considered for re-scheduling and submission in the next iteration of the algorithm. Recall that the adoption of the knapsack formulation has been made because it is not possible to assign all the tasks for execution to the Grid, since the total workload exceeds the capacity of the Grid infrastructure. Postponing the execution of a task may result, in some cases, to a deviation from deadline. This deviation can also be handled by other mechanisms, such as in [48], so as to guarantee a specific level of QoS. The concept of the study in [48] is the adoption of a *max–min fair share* algorithm to manage the assignment of tasks on the Grid resources. The results in that study show that it is possible to distribute the computational rate of the Grid infrastructure in a fair manner to each task, serving as many replicas as is possible in order to maintain a specific fault tolerance level to the system, but regardless of the profit.

5.2. Comparison of algorithms for the 0/1 knapsack problem

All versions of the knapsack problem belong to the family of NP-hard problems, meaning that no polynomial algorithm

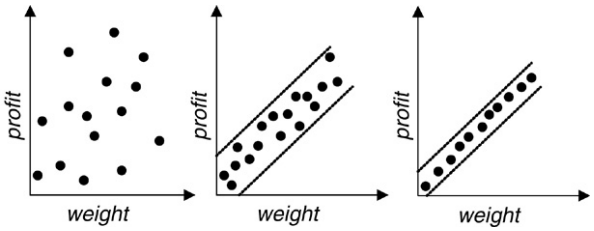


Fig. 3. Uncorrelated, weakly correlated and strongly correlated data examples.

has been devised until now for these problems. Therefore, to find the optimal solution, an enumeration of the solution space is required. However, substantial effort may be saved by employing techniques like branch-and-bound, dynamic programming, state space relaxation, and preprocessing [47]. Overviews of exact solution techniques are also presented in [49] and [50]. Moreover, heuristics that generate feasible, and usually suboptimal, solutions within short execution times can be found in [50,51]. Ref. [52] presents an algorithm for generating near-optimal solutions to the 0–1 knapsack problem, with bounds on the sub-optimality of the solution output, by embedding a local search based heuristic procedure within a branch-and-bound framework. Greedy heuristics are based on placing items in order of their highest to lowest profit-to-weight ratios and then attempting to fill the knapsack in that sequence. These can be used to initially fill the knapsack, and then simple local search heuristics — such as taking one item out of the knapsack and replacing it with a different item — can be used to improve the initial solution. Also, heuristic approaches based on genetic algorithms have been investigated [53–55].

The performance of the algorithm depends on the nature of the data instances considered in the specific application. The data instances may be the following (see Fig. 3). (i) *Uncorrelated*: where no correlation exists between the profit and the weight of the items. These instances are generally easier to solve, since (a) the large variance among the weights makes it easier to obtain a filled knapsack and (b) it is easier to eliminate numerous items a priori by applying upper bound tests. (ii) *Weakly correlated*. In these instances, there is some degree of correlation between the profit and the weight. Due to the existence of correlation, it is difficult to eliminate items by upper bound tests. However, the large variation in the weights makes it easier to obtain a filled knapsack and, due to the correlation, filled solutions are generally very close to the optimal solution. (iii) *Strongly correlated*: In these instances, the profit of an item is a linear function of its weight. Strongly correlated instances are the hardest cases to solve, and therefore they are often used as a measure of an algorithm's ability to solve difficult problems.

In the case of tasks submitted to the Grid for execution, the degree of correlation between the weight and the value of each task depends on the adopted pricing and charging policy. In most cases, it is reasonable to assume that we deal with weakly correlated data, since: (a) a higher task weight means higher consumption of grid resources and thus a higher price charged for the execution of the task, and (b) the price charged for a task may also be subject to parameters other than the resource usage (e.g. in an emergency situation, a heavy task may receive little or no charge, a client may be willing to pay a high price for a light task if it is of critical importance to him, etc.) Generally, it is a subject for further research to evaluate and adopt pricing schemes for the submitted jobs in mobile Grids, something that will determine the correlation between the workload of the submitted tasks and the profit that will be gained. This is due to contradictory requirements which reflect unpredictable changes of the on-site demand of aggregated computational power and the unreliable links that may occur. This generally leads to complex and dynamic schemes which would result in uncorrelated instances for the given algorithms.

In our study, for the solution of the knapsack problem, we have considered four techniques: *Backtracking (BT), Branch and Bound (BB), Dynamic Programming (DP)*, and a *heuristic greedy* algorithm *(GR)*. The first three techniques are based on a systematic examination of the list of all candidate solutions, which achieves significant run-time savings in both the worst and expected cases [56]. In backtracking, the solution space is first organized in a tree structure so that it can be searched easily. Then the solution space is searched in a depth-first manner, beginning at a start node, using bounding functions to avoid moving into subspaces that lead to solutions that are either infeasible or impossible to be the answer. Like backtracking, branch-and-bound also organizes the solution space in a tree structure, but a breadth-first or least-cost search is performed instead. Branch-and-bound is presented in the literature as finding the optimal solution a bit sooner than backtracking [56]. However, branch-and-bound algorithms have more space requirements than backtracking, resulting in successful answers where memory limitations are applicable.

Dynamic programming is frequently used when a recursive solution to a problem has exponential run time, due to repeated computation. The technique used is to store the results of particular calls of a method or computation that have already been computed, using a table indexed by the different argument values. In general, this technique is known as "memoization". Although this technique is very useful, it can potentially require a large amount of space, because of creating arrays big enough to cover all possible argument values. Dynamic programming involves the use of a memo that stores past results, but also involves changing the order of computation, bringing the memo to the fore by systematically building up and recording sub-problem results, to provide a time- and space-efficient overall solution.

As the number of tasks increases, the run time required for the total enumeration of the solution space and the selection of the optimal one can be prohibitive. A *greedy* solution strategy can be employed in these cases in order to find a "near-optimal" solution. In this approach, the items are placed in the knapsack in a "one-by-one" basis and, once an item has been selected, it cannot be removed. A greedy algorithm for an optimization problem always makes the choice that looks best at the moment and adds it to the current sub-solution. For the 0–1 knapsack problem, the most common criteria that can be used to select the next item to be placed into the knapsack are the following. (i) Profit: At each step, select from the remaining items the one

with the highest profit (without exceeding the total capacity of the knapsack). This approach tries to maximize the profit by choosing the most profitable items first. (ii) Weight: At each step, select from the remaining items the one with the least weight (provided that the capacity of the knapsack is not exceeded). This approach tries to maximize the profit by putting as many items into the knapsack as possible. (iii) Profit density: At each step, select from the remaining items the one with the largest profit density (p/w) (provided that the capacity of the knapsack is not exceeded). This approach tries to maximize the profit by choosing items with the largest profit per unit weight [57]. The latter is the one used for the evaluation of the proposed scheme in Section 6.

Even in the case where we have no estimation of the tasks' profits, or all the tasks are of equal profit, the problem formulation is again based on the 0–1 knapsack problem, where now the objective is to maximize the number of selected tasks, searching for those that have less workload. If the number of tasks allows exhaustive enumeration of solution space, then we use backtracking or branch-and-bound to find all feasible ways to fill the knapsack and we select the one with the maximum number of items.

Table 1
Calculation of the number of necessary replicas m_i in order to achieve the desired fault tolerance threshold λ, assuming the special case of having constant failure probability g_i for each Grid resource

g_i	$\lambda = 0.01$		$\lambda = 0.05$		$\lambda = 0.10$	
	$\frac{\log(\lambda)}{\log(g_i)} - 1$	m_i	$\frac{\log(\lambda)}{\log(g_i)} - 1$	m_i	$\frac{\log(\lambda)}{\log(g_i)} - 1$	m_i
0.01	0.000	0	−0.349	0	−0.500	0
0.02	0.177	1	−0.234	0	−0.411	0
0.03	0.313	1	−0.146	0	−0.343	0
0.04	0.431	1	−0.069	0	−0.287	0
0.05	0.537	1	0.000	0	−0.231	0
0.06	0.637	1	0.065	1	−0.182	0
0.07	0.732	1	0.127	1	−0.134	0
0.08	0.823	1	0.186	1	−0.088	0
0.09	0.912	1	0.244	1	−0.044	0
0.10	1.000	1	0.301	1	0.000	0
0.15	1.427	2	0.579	1	0.214	1
0.20	1.861	2	0.861	1	0.431	1
0.25	2.322	3	1.160	2	0.661	1
0.30	2.825	3	1.488	2	0.912	1
0.35	3.387	4	1.854	2	1.193	2
0.40	4.026	5	2.269	3	1.513	2
0.45	4.767	5	2.752	3	1.887	2
0.50	5.644	6	3.322	4	2.322	3

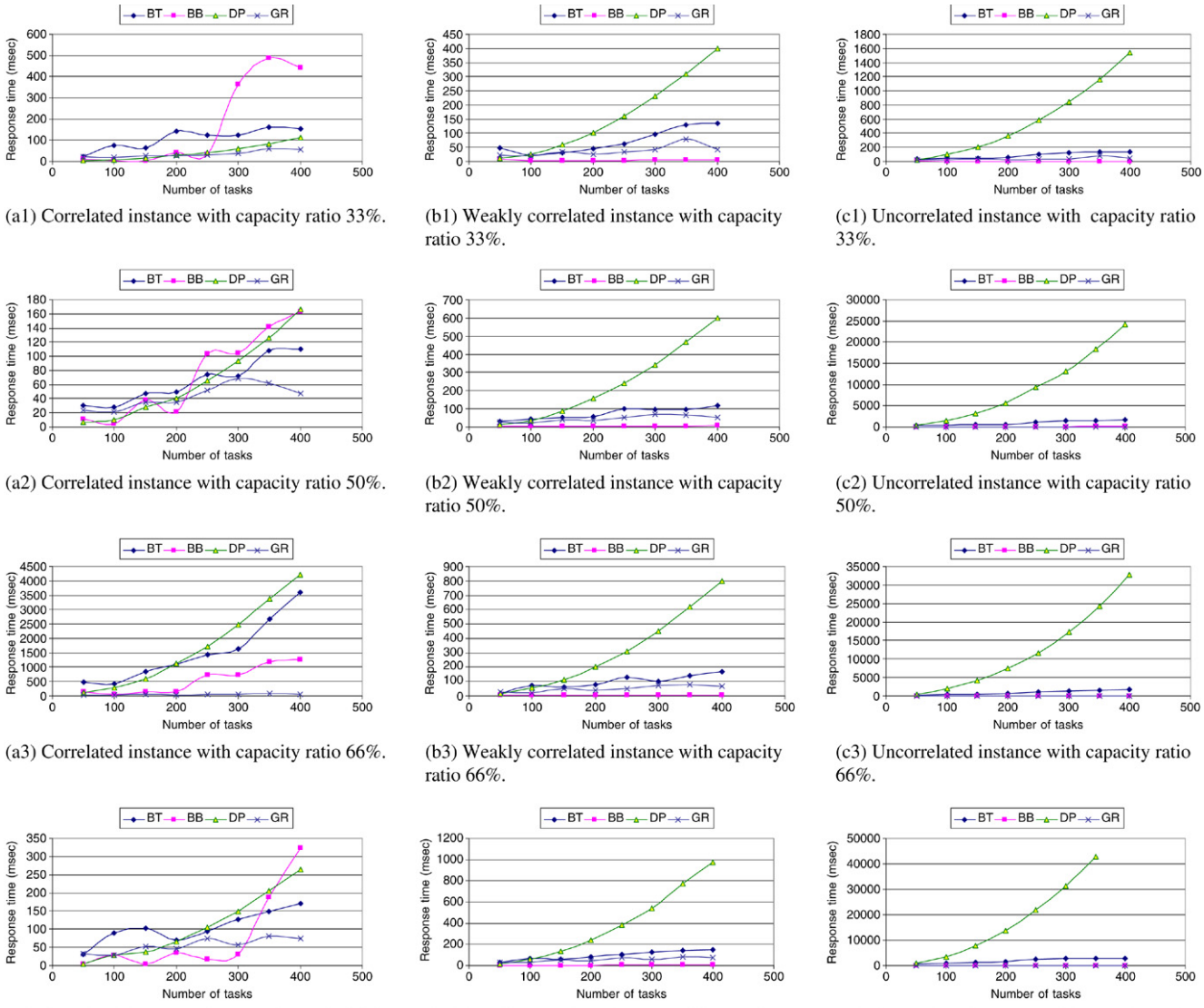
6. Simulation results

6.1. Evaluation of the knapsack algorithms

The proposed scheme has been implemented and evaluated through simulation results. In order to be able to guarantee a desired fault tolerance level for the mobile Grid, we have to identify the fault tolerance threshold, λ. Based on experimental results and on general practices from real-world applications, a satisfactory level of fault tolerance in such a distributed environment with unreliable wireless links and mobile devices (normally prone to omissions and energy limitations) has to be increased enough in order to allow a normal operation. Current practices in various Service Level Agreement contracts imply a continuous operation of the services provided by their infrastructure, which is in total about 99% of the total time. As we have already mentioned, it is important to identify (based on the reliability of the individual Grid resources) the number of replicas that have to be produced and scheduled for execution. For various values of a fault tolerant threshold, we obtain the number of replicas according to Eq. (18) as presented in Table 1.

While evaluating the system through simulation results, we defined a virtual mobile Grid environment of a specific computational rate μ_G. In order to identify the instances used for the various algorithms, we defined the quantity of *capacity ratio*, which represents the ratio of the total workload of all the tasks that are to be submitted for execution to the total capacity of the Grid over the given period. The capacity ratio (CR) can be represented as $CR = W/C$ and practically represents the relation between the demand for computational capacity of the Grid resources (expressed by the user task submissions) and the capability of the Grid infrastructure to accept and execute these tasks (practically its capacity over this given period). Although the bigger this capacity is the more tasks it can execute, there is no commercial oriented system that can undertake successfully 100% of its user requests at the same time. It is therefore mandatory to evaluate such a resource management scheme in a manner that would make visible its operation in various load instances. Especially in mobile Grids, which are very dynamic and subject to topological changes, it is possible to have peaks of computational capacity demand more often in given periods. For this reason, the proposed model has been stressed for instances up to $CR = 33\%$. Fig. 4 presents an overview of the results that have been derived for the evaluation of the selected algorithms, regarding the time needed by each algorithm in each case to select the tasks to submit for execution.

From Fig. 4 we can see the experimental behavior of the adopted algorithms in various instances of correlation "weight–profit" for the given numbers of tasks and the capacity ratio of the simulated Grid. The presented instances provide the explanation and motivation of why different algorithms have been used for the solution of the knapsack formulated problem, while their theoretical behavior is well known from the literature. It can be seen that the DP with "memoization" shows an increased complexity in terms of time (and space in memory, due to the multiple intermediate results that it has to keep). This algorithm shows more stable behavior, regardless of the correlation instance and the capacity ratio. However, this stability, which can be beneficial for prediction purposes and QoS guarantees, has the drawback that the space complexity caused by the Grid capacity may prohibit the use of this algorithm for very large Grid infrastructures (something which can be avoided in mobile and dynamic environments). On the other hand, we can see that, for the instance of having correlated weight–profit pairs for the tasks and a heavy loaded Grid infrastructure (or equivalently a Grid with small capacity that



(a1) Correlated instance with capacity ratio 33%.
(b1) Weakly correlated instance with capacity ratio 33%.
(c1) Uncorrelated instance with capacity ratio 33%.
(a2) Correlated instance with capacity ratio 50%.
(b2) Weakly correlated instance with capacity ratio 50%.
(c2) Uncorrelated instance with capacity ratio 50%.
(a3) Correlated instance with capacity ratio 66%.
(b3) Weakly correlated instance with capacity ratio 66%.
(c3) Uncorrelated instance with capacity ratio 66%.
(a4) Correlated instance with capacity ratio 80%.
(b4) Weakly correlated instance with capacity ratio 80%.
(c4) Uncorrelated instance with capacity ratio 80%.

Fig. 4. Comparison of the four algorithms (Backtracking (BT), Branch and Bound (BB), Dynamic Programming (DP), and the Greedy algorithm (GR)) concerning the response time for the derived solution over various instances of the *number of tasks* and the Grid capacity. Column (a1)–(a4) presents the correlated instance between the weight and the profit of the tasks for capacity ratios of 33%, 50%, 66% and 80%, respectively. Weakly correlated and uncorrelated instances for the same four different capacity ratios are presented in the columns (b1)–(b4) and (c1)–(c4).

has to execute a significantly greater workload than the one it can have), the DP algorithm (in case Fig. 4(a1)) performs better than the other two. The correlated instance is the harder one for all the algorithms (as has already been mentioned), since it is difficult for the algorithms to decide whether to take a specific task into the knapsack. The DP with memoization, however, outperforms BB and BT, since it uses the intermediate results to make quicker decisions. However, it should be mentioned that the time required for such calculations is very small, which is suitable for on-line usage in Grid middleware.

As the various correlation instances tend to be "weakly" and "un-" correlated, for every case of capacity ratio we observe that BB and BT present a more or less similar curve (BB appears to be more efficient in terms of time) and they outperform the DP algorithm. This is clearer particularly in Fig. 4(c4), where BB is slightly faster than BT, but they have a significant difference with DP — something that would prohibit the usage of the latter for such instances. But what explains the performance of BB (and BT) is the philosophy of the algorithms to totally exclude "branches" of no good solutions (which can be the case for weakly or uncorrelated instances, where we can have cases of very heavy tasks with very small profit) and thus taking quicker decisions on which tasks to load on the Grid resources.

The heuristic greedy algorithm, which uses the profit density as a criterion for the selection of the tasks to be put in the knapsack, presents significantly better performance in terms of response time. The drawback in this case is the fact that, since it uses a heuristic method for the filling of the knapsack, it does not necessarily conclude with the overall optimal value — something that is the case for the three other algorithms. We measured the deviation from the optimal value for a given instance of Fig. 4(c4) and we obtained the results presented in

Table 2
Comparison between the *optimal value* for the profit gained from the BB algorithm (measured in *currency units*) and the value achieved as a *solution* of the *Greedy* algorithm for 10 different inputs of 300 tasks in a Grid infrastructure with $CR = 80\%$, and for the three different correlation instances

Correlated instance		Weakly correlated instance		Uncorrelated instance	
Optimal value (currency units)	Greedy solution (currency units)	Optimal value (currency units)	Greedy solution (currency units)	Optimal value (currency units)	Greedy solution (currency units)
61 111	61 067	61 482	61 461	60 498	60 481
60 917	60 701	61 203	61 102	60 266	60 162
61 052	61 045	61 499	61 430	59 373	59 357
61 082	61 045	60 699	60 469	60 058	60 005
61 026	61 019	60 982	60 904	61 843	61 760
61 188	60 950	60 024	59 914	61 379	61 274
61 119	61 110	63 074	63 063	62 693	62 638
61 157	60 926	63 159	63 155	60 177	60 055
61 076	60 839	63 214	63 094	61 578	61 499
60 685	60 648	63 932	63 834	59 565	59 527

Table 2. As we can see, the difference between the *optimal values* achieved through BB, BT and DP and the heuristic *greedy solution* value for the case of having 300 tasks with a Grid capacity ratio of 80% is almost always present, but within specific limits which advance the usage of such an algorithm for the very large amounts of tasks to be scheduled. When the profit is measured in *currency units* that can be achieved as optimal values of BB, BT and DP for correlated, weakly correlated and uncorrelated instances, the greedy heuristic provides a solution ranging from 99.6% to 99.9% of the optimal solution. If we consider that the greedy algorithm does not have exponential behaviour with respect to the input size of the problem (number of tasks), it can be an ideal solution to be used for very large-scale Grid systems with many task submissions per unit time. From the perspective of commercial exploitation, the profit gained from the greedy algorithm is less than the optimal one, which can be derived using the BB algorithm (for example). In the case of having a small set of tasks (i.e. 50 or 100 tasks), the difference in time is very small and can be balanced with the additional profit that can be achieved with this algorithm. So, the usage of such an algorithm would be more beneficial for the Grid service provider.

6.2. Evaluation of the overall replication scheme

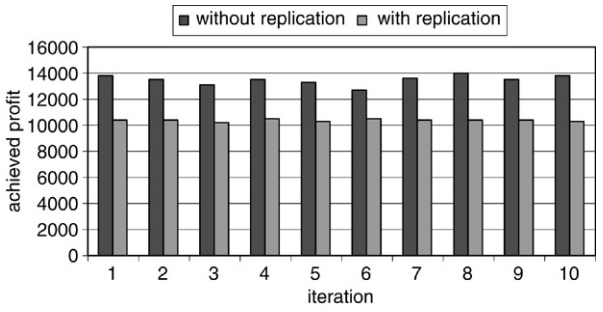
A set of $N = 300$ tasks has been created for scheduling in a mobile Grid environment. The tasks follow a weakly correlated model, having a profit variance between 10 and 100 *currency units*. We associate a failure probability between specific values for each task. This failure probability (as already described in previous sections) is derived using the failure probability model of the resource, where the task is going to be executed. We have considered two different approaches. The first is to use the knapsack scheme only on the primary tasks without applying the replication mechanism, while the second involves the replication procedure, as has been described in Section 4.2. The tasks have been assigned for execution on specific resources, and resource failure has been applied (based on the proposed model) so as to have a realistic scenario of task failures in this context. We examined the efficiency of the system in terms of the number of successful tasks (which is the number of tasks that have been selected to be loaded in the knapsack and have finished successfully) and the achieved profit (which is the profit associated with these tasks). Recall that we do not use any "pricing" scheme for adapting the task profit to the new weight caused by the additional workload of the replicas. For each individual case, we repeat the execution 10 times, so as to examine the system for having diverse task failures for the given set of tasks. The simulated results are presented in Fig. 5.

We observe that the variation in profit for the case of filling the knapsack with tasks that have not been replicated is higher than that in the case of the replicated tasks. This is due to the fact that the replicated tasks minimize the failure probability and provide a more guaranteed result, based on the fault tolerance threshold of the Grid environment. We have to mention that the total achieved profit in the case of having no replicas is higher since, for a given workload (weight), we receive a specific profit, while in the case of replicas we get the same profit for multiple times of its workload. The results that are presented are interesting, since they show the relation of the profit to the fault tolerance threshold, the reliability of the resources, and the capacity ratio of the Grid. Moreover, although one would expect that such an increase in the workload would have a significant impact on the reduction of the profit gained, we see that the guarantee of having successful executions in unreliable environments is beneficial.

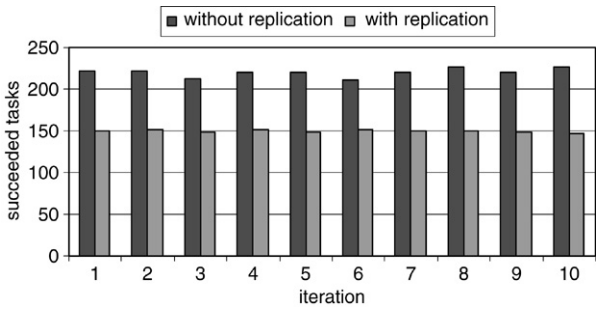
In particular, in the case of having a desired overall reliability of 91% ($\lambda = 0.09$), a capacity ratio of 80%, and highly unreliable resources, the achieved profit in the case of replication is very close to that in the case of not having replicas (with a minimum deviation of 9%). For the cases presented in Table 2, we have a mean deviation of about 16% for the achieved profit for both methods, namely of having replication and not having replication.

Another important remark is that the deviation of the achieved profit from the total profit in the knapsack in the case of applying replication ranges from 1.41% to 7.27%, while, in the case of not having replication, this deviation reaches 28.63% (when we have highly unreliable Grid resources).

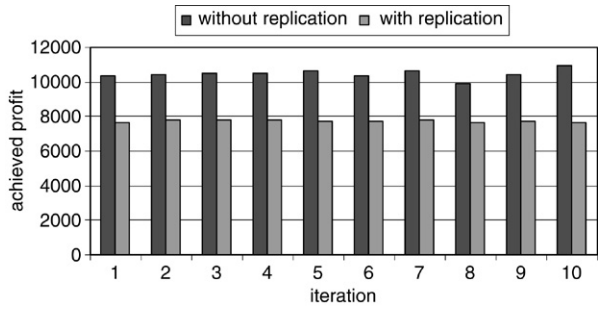
The non-replication case usually presents a higher profit and number of successful tasks than the case where we have



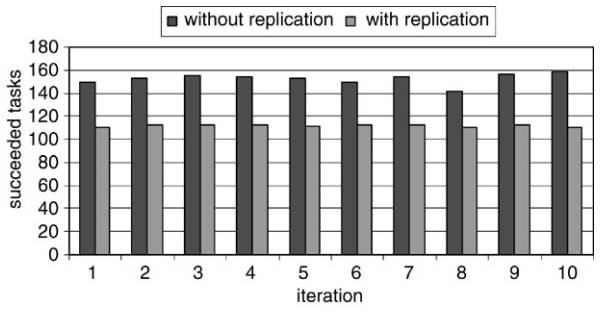
(a1) $\lambda = 0.03$, P_f in [0.05–0.30], $CR = 80\%$.



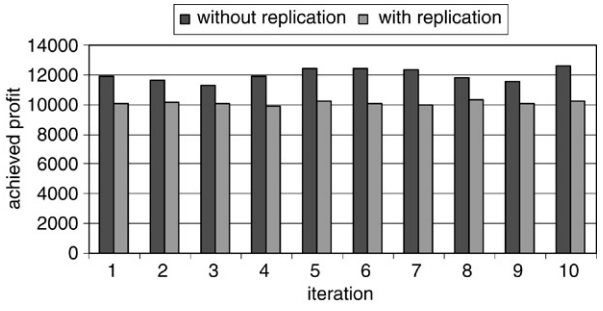
(a2) $\lambda = 0.03$, P_f in [0.05–0.30], $CR = 80\%$.



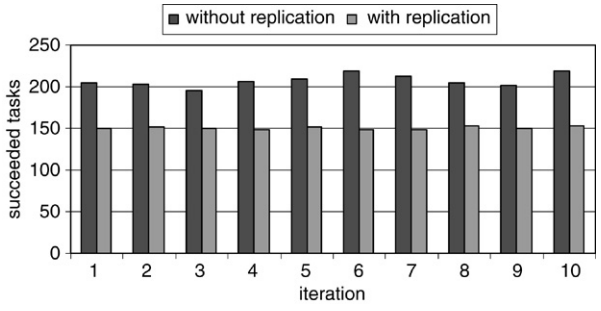
(b1) $\lambda = 0.03$, P_f in [0.05–0.30], $CR = 50\%$.



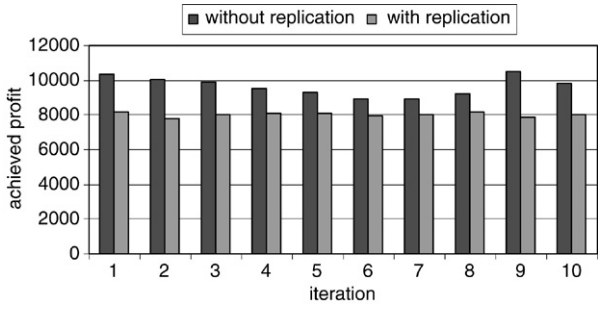
(b2) $\lambda = 0.03$, P_f in [0.05–0.30], $CR = 50\%$.



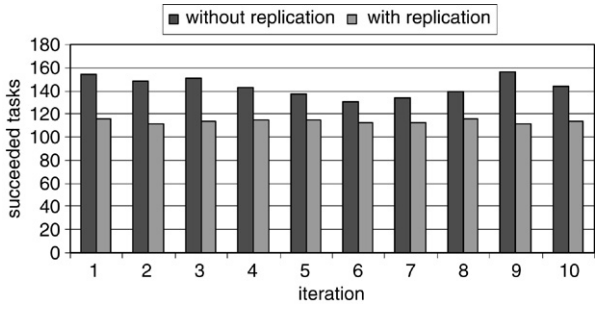
(c1) $\lambda = 0.06$, P_f in [0.10–0.35], $CR = 80\%$.



(c2) $\lambda = 0.06$, P_f in [0.10–0.35], $CR = 80\%$.



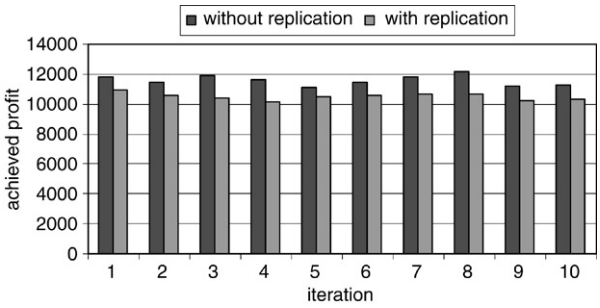
(d1) $\lambda = 0.06$, P_f in [0.10–0.35], $CR = 50\%$.



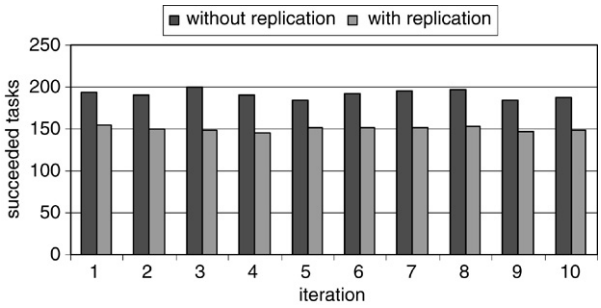
(d2) $\lambda = 0.06$, P_f in [0.10–0.35], $CR = 50\%$.

Fig. 5. Simulation results of the proposed scheme. Column 1 ((a1)–(f1)) shows the *achieved profit* of the successful tasks for both techniques "with replication" and "without replication". The iteration in the horizontal axis of the figures means multiple execution of the same instance in order to examine the failure of the tasks. Column 2 ((a2)–(f2)) gives the number of the *successful tasks* for both techniques. Each figure gives the instance at which it refers to in terms of fault tolerance threshold, capacity ratio and the interval of the tasks failure probabilities.

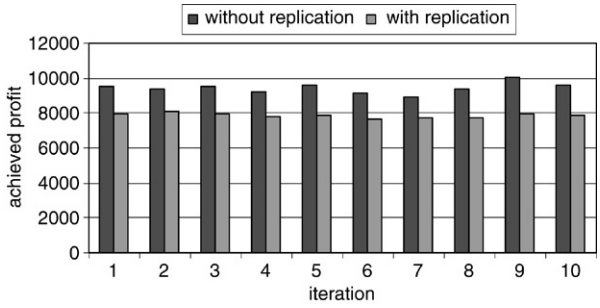
replication, as can be seen in Table 3. In the non-replication case, a significant number of tasks that were initially scheduled in the system fail to complete execution. These failures result in not receiving the profit that is associated with the successful execution of the concrete task. This is not to mention other costs that may be applied which are related to deadline violations, Service Level Agreement (SLA) violations, etc., and which are not examined in this paper. In this way, the actual profit



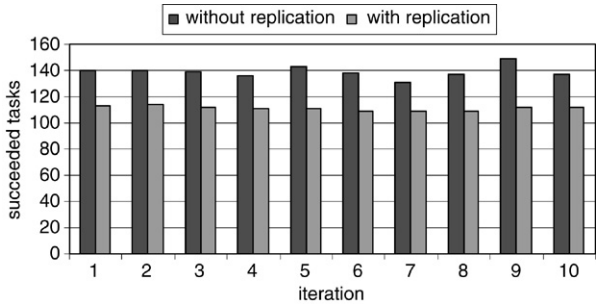
(e1) $\lambda = 0.09$, P_f in [0.15–0.40], $CR = 80\%$.



(e2) $\lambda = 0.09$, P_f in [0.15–0.40], $CR = 80\%$.



(f1) $\lambda = 0.09$, P_f in [0.15–0.40], $CR = 50\%$.



(f2) $\lambda = 0.09$, P_f in [0.15–0.40], $CR = 50\%$.

Fig. 5. (*continued*)

Table 3
Overview of the results for the simulation procedure

$N = 300$			Total tasks in the knapsack (a)	Successful tasks (b)	Total profit of tasks in knapsack (c)	Achieved profit (from the successful tasks) (d)	Ratio of achieved profit (e)	Mean P_f in the knapsack (f)
$\lambda = 0.03$	$CR = 80\%$	Without replication	264	220	16 174	13 483	0.83	0.18
		With replication	152	150	10 532	10 383	0.99	0.01
P_f [0.05, 0.30]	$CR = 50\%$	Without replication	188	153	12 918	10 471	0.81	0.18
		With replication	113	111	7 854	7 730	0.98	0.01
$\lambda = 0.06$	$CR = 80\%$	Without replication	265	207	15 183	11 984	0.79	0.22
		With replication	155	151	10 417	10 112	0.97	0.03
P_f [0.10, 0.35]	$CR = 50\%$	Without replication	188	144	12 683	9 661	0.76	0.23
		With replication	117	114	8 257	8 029	0.97	0.03
$\lambda = 0.09$	$CR = 80\%$	Without replication	269	192	16 243	11 592	0.71	0.28
		With replication	157	150	10 979	10 516	0.96	0.05
P_f [0.15, 0.40]	$CR = 50\%$	Without replication	190	139	12 882	9 434	0.73	0.27
		With replication	120	111	8 477	7 860	0.93	0.06

The table presents a comparison between the replication and no-replication approach for various instances of fault tolerance threshold, failure probabilities of the tasks, and capacity ratios. Column (a) gives the number of selected tasks that are in the knapsack. (b) gives a simulation result for specific tasks that have succeeded. In column (c) we have the total profit of the tasks in the knapsack, where we see a visible difference between the two methods "with replicas" and "without replicas". In (d) we show the achieved profit from the tasks in the knapsack that have been successfully executed. Column (e) shows the ratio of the achieved profit from the total profit rounded to the first two digits. Column (f) shows the mean failure probability of the tasks in the knapsack where, in the case of having no replicas, it represents the task itself, while, in the case of replication, it represents the product of the failure probabilities of the original task and its replicas.

that may be achieved could be even less. The interesting aspect of the results presented in Table 3 is that, in the cases of replication, the number of successfully executed tasks, as well as their achieved profit, is very close to those initially scheduled in the system (which are being contained in the knapsack). They are also close enough to the achieved profit that we get (for the same instances) in the cases of not having replication. This means that, by providing fault tolerance in highly unreliable environments, with the proposed scheme we can achieve a profit that is comparable to the case of having non-replication. Moreover, by using replication, we enhance the mobile Grids with reliability which allows for respecting the deadlines set by the users and guaranteeing QoS that satisfies the customers' requirements. The proposed replication scheme

is especially suitable for cases where the successful execution of tasks is important in unreliable environments (e.g. critical situations, deadlines that have to be met, etc.). In those cases, the benefit from minimizing the failures overcomes the trade-off for allowing some resources to be occupied by task replicas rather than accepting more prototype tasks in the system.

7. Conclusions and discussion

In this paper we have studied a fault tolerant model for mobile Grid environments based on the task replication concept. The tasks are replicated on the basis of their failure probability and the desired fault tolerance level. The estimation of the failure probability for a task that has been scheduled to a Grid resource for execution is based on the Weibull reliability model. Since the additional replicas produce an overhead on the workload stressed to the Grid's total computational capacity, we introduce an additional mechanism, based on the knapsack formulation, for managing the tasks and their replicas in terms of their scheduling, so as to maximize the Grid resources utilization in terms of the achieved profit. For this model, four different algorithms have been used (Backtracking, Branch and Bound, Dynamic programming with "memoization", and a greedy algorithm based on the optimization of profit density). The model has been implemented and evaluated for a variety of tasks with a diverse set of failure probabilities and their replicas and with various instances for correlation between the 'weight' of the tasks and the profit that their execution would result in a commercial oriented Grid. The results have indicated the efficiency of the proposed scheme for the response time, Grid resource utilization and profit maximization, and are promising for future research on this topic.

The main contribution of this work lies in the analysis of fault tolerance in mobile Grids using an efficient task replication scheme that is designed for diverse failure probabilities of the resources and aims to operate in the Grid middleware, regardless of the scheduling policy. Our study has contributed to the formulation of the Grid resource management topic as a knapsack problem and has simulated its solution through specific algorithms that are well known in the relative domain, and has shown that their usage for similar scenarios can improve the overall utilization of the resources as a result of minimizing task failures (see Sections 6.1 and 6.2). It has examined the relation between the desired fault tolerance threshold on a mobile Grid environment, the instance of correlation for the weight–profit of the submitted tasks, and the capacity capability of a given Grid infrastructure for the design of commercially viable mobile Grids. The paper focuses on the mobile Grid concept by taking into consideration the main attributes of a typical Grid system. Although many of the principles that are mentioned are also met in traditional real-time distributed systems, or in high-performance and parallel computing, in this study they concern a middleware approach that extends the basic functionality for performance optimization. What is typically Grid-like in this study is the ability of the proposed scheme to provide support for dynamic and heterogeneous environments, and to also enable, besides performance optimization, a self-manageability framework through general-purpose and simple mechanisms. Using this approach, we can have a simple-to-develop mechanism based on the specific requirements for mobile grids, which can enhance the Grids' architectural aspects and provide Quality of Service features to highly unreliable infrastructures.

Finally, it enables the discussion of a variety of issues for future research in the domain of Grid fault tolerance and economics. An interesting aspect is to deploy other modifications of these algorithms (for instance, stochastic dynamic programming, or other heuristics) that show better theoretical results in the literature [49–51]. Although more complex to implement and manage, it would be interesting to see the difference in their performance under experimental conditions. Given the fact that the current trend in mobile Grid computing imply the adoption of the OGSA [58] based architectural design implemented by WSRF [59], it is interesting to translate the task replication and submission schemes into the replication of requests as Simple Object Access Protocol (SOAP) [60] messages towards services that would be running on the mobile servers. While having failures in the resources of such a dynamic environment, it is interesting to show also how these changes (in topology and computational capability of the Grid) affect the efficiency in terms of deviation from deadline for the tasks. We can assume that the failure to execute a task (or at least one of its replicas) results in a cost q_i for the system. This cost may be due to the consumption of resources or the price (penalty) that has to be paid to the user due to deadline and SLA violation or the discontent caused to the client (harming the company's image) etc. It is an interesting topic to examine how, in such a dynamic environment, a viable economic model could be developed for estimating the cost of the replicas so that not only the 'weight' of the total task increases but also the profit that will be gained in an adjusted manner, so that the total profit will be guaranteed, as if the grid had only reliable resources. The idea of preserving the QoS attributes of the system adds an important aspect to various business models, because of the static QoS information that can then be integrated into SLAs. This can contribute to simplifying the dynamic SLA mechanisms and, in turn, the dynamic workflow techniques, which are currently one of the major concerns of the Grid scientific community. Finally, a hybrid model could be designed and developed that would estimate the expected profit and decide which strategy would be ideal to follow in terms of performance and profit optimization.

Acknowledgments

This work has been partially supported by the AKOGRIMO Integrated Project (FP6-2003-IST-004293) [19]. The authors would like to thank all the anonymous reviewers for their constructive and fruitful comments, which helped in improving the quality, presentation and organization of the paper.

References

[1] M.R. Lyu, Software Fault Tolerance, John Wiley & Sons, Chichester, UK, 1995.

[2] A. Nguyen-Tuong, Integrating fault-tolerance techniques in Grid applications, Ph.D. Dissertation, University of Virginia, August 2000.
[3] K. Ramamritham, J.A. Stankovic, P.-F. Shiah, Efficient scheduling algorithms for real-time multiprocessor systems, IEEE Trans. Parallel Distrib. Syst. 1 (2) (1990) 184–194.
[4] Scheduling Working Group of the Grid Forum, Document: 10.5, September 2001.
[5] F. Wang, K. Ramamritham, J.A. Stankovic, Determining redundancy levels for fault tolerant real-time systems, IEEE Trans. Comput. 44 (February) (1995).
[6] D. Pisinger, Algorithms for Knapsack problems, Ph.D. Thesis, Dept. of Computer Science, University of Copenhagen, February 1995.
[7] J.B. Weissman, Fault Tolerant Computing on the Grid: What are My Options? HPDC (1999).
[8] S. Hwang, C. Kesselman, A flexible framework for fault tolerance in the grid, J. Grid Comput. 1 (2003) 251–272.
[9] The Globus project, http://www-fp.globus.org/hbm/.
[10] A. Nguyen-Tuong, A.S. Grimshaw, Using reflection to incorporate fault-tolerance techniques in distributed applications, Computer Science Technical Report, University of Virginia, CS 98-34, 1998.
[11] S.J. Chapin, D. Katramatos, J. Karpovich, A. Grimshaw, Resource management in Legion, Future Gener. Comput. Syst. 15 (5–6) (1999) 583–594.
[12] H. Casanova, J. Dongarra, C. Johnson, M. Miller, Application-specific tools, in: I. Foster, C. Kesselman (Eds.), The GRID: Blueprint for a New Computing Infrastructure, 1998, pp. 159–180 (Chapter 7).
[13] J.S. Plank, H. Casanova, M. Beck, J.J. Dongarra, Deploying fault tolerance and task migration with NetSolve, Future Gener. Comput. Syst. 15 (5) (1999) 745–755.
[14] A.S. Grimshaw, A. Ferrari, E.A. West, Mentat, in: G.V. Wilson, P. Lu (Eds.), Parallel Programming Using C++, 1996, pp. 382–427 (Chapter 10).
[15] F.C. Gartner, Fundamentals of fault-tolerant distributed computing in asynchronous environments, ACM Comput. Surv. 31 (1) (1999).
[16] Q. Chen, M. Ferris, J.T. Linderoth, FATCOP 2.0: Advanced features in an opportunistic mixed integer programming solver, Ann. Oper. Res. 103 (2001) 17–32.
[17] H. Zhuge, X. Sun, J. Liu, E. Yao, X. Chen, A scalable P2P platform for the knowledge grid, IEEE Trans. Knowl. Data Eng. 17 (12) (2005) 1721–1736.
[18] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: Proc. 16th Int'l Conf. Supercomputing, June 2002, pp. 84–95.
[19] Access to Knowledge through the Grid in a Mobile World (AKOGRIMO) Integrated Project FP6-2003-IST-004293, http://www.akogrimo.org/.
[20] A. Litke, D. Skoutas, T. Varvarigou, Mobile Grid computing: Changes and challenges of resource management in a mobile grid environment, in: Access to Knowledge through the Grid in a Mobile World Workshop, held in conjunction with 5th Int. Conf. on Practical Aspects of Knowledge Management, PAKM 2004, Vienna. Available at http://www.mobilegrids.org/docs/pakm2004/papers/3_Mobile-Grid-Computing.pdf.
[21] D. Abramson, R. Sosic, J. Giddy, B. Hall, Nimrod: A tool for performing parametised simulations using distributed workstations, in: The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
[22] D. Abramson, R. Buyya, J. Giddy, A computational economy for grid computing and its implementation in the Nimrod-G resource broker, Future Gener. Comput. Syst. 18 (8) (2002) 1061–1074.
[23] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolski, The GrADS Project: Software support for high-level grid application development, Int. J. High Perform. Comput. Appl. 15 (4) (2001) 327–344. Winter.
[24] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke, Condor-G: A computation management agent for multiinstitutional grids, Cluster Comput. 5 (2002) 237–246.
[25] M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, D. Zagorodnov, Adaptive computing on the grid using AppLeS, IEEE Trans. Parallel Distrib. Syst. 14 (4) (2003) 369–382.
[26] C. Weng, X. Lu, Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid, Future Gener. Comput. Syst. 21 (2005) 271–280.
[27] V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, in: Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland, July 2002, pp. 359–367.
[28] The Unicore project, http://www.unicore.org/forum.htm.
[29] R. Parra-Hernandez, D. Vanderster, N.J. Dimopoulos, Resource management and Knapsack formulations on the grid, in: Proc. of the 5th IEEE/ACM Int. Workshop on Grid Computing, GRID'04.
[30] J.Y.-T. Leung, M.L. Merrill, A note on preemptive, scheduling of periodic, real-time tasks, Inform. Process. Lett. (November) (1980) 115–118.
[31] L.E. Jackson, G.N. Rouskas, Deterministic preemptive scheduling of real time tasks, IEEE Comput. 35 (5) (2002) 72–79.
[32] A.S. Tanenbaum, M. van Steen, Distributed Systems: Principles and Paradigms, 1st ed., Prenctice Hall, Computer Science.
[33] T. Varvarigou, J. Trotter, Module replication for fault-tolerant real-time distributed systems, IEEE Trans. Reliab. 47 (1) (1998) 8–18.
[34] D.A. Reed, C. Lu, C.L. Mendes, Reliability challenges in large systems, Future Gener. Comput. Syst. 22 (2006) 293–302.
[35] D. Nurmi, J. Brevik, R. Wolski, Modeling machine availability in enterprise and wide-area distributed computing environments, UCSB Computer Science Technical Report Number CS2003-28.
[36] D.C. Montgomery, G.C. Runger, Applied Statistics and Probability for Engineers, 3rd ed. (an Interactive e-text).
[37] P.L. Meyer, Introductory Probability and Statistical Applications, 2nd ed., Addison-Wesley, 1970 (Chapter 11).
[38] R.L. Scheaffer, Introduction to Probability and Its Applications, 2nd ed., Duxbury, 1995 (Section 4.9).
[39] R.B. Abernethy, The New Weibull Handbook, 4th ed., ISBN 0-9653062-1-6.
[40] N. Doulamis, A. Doulamis, A. Panagakis, K. Dolkas, T. Varvarigou, E. Varvarigos, A combined fuzzy-neural network model for non-linear prediction of 3D rendering workload in grid computing, IEEE Trans. Syst. Man Cybern. B. (2004).
[41] X. He, X.-H. Sun, Gregor von Laszewski, QoS guided min-min heuristic for grid task scheduling, in: Grid Computing, J. Comput. Sci. Technol. 18 (4) (2003) (special issue).
[42] L. Carrington, A. Snavely, N. Wolter, A performance prediction framework for scientific applications, Future Gener. Comput. Syst. 22 (2006) 336–346.
[43] R. Wolski, N. Spring, J. Hayes, The network weather service: A distributed resource performance forecasting service for metacomputing, Future Gener. Comput. Syst. 15 (1999) 757–768.
[44] L. Gong, X.H. Sun, E. Waston, Performance modeling and prediction of non-dedicated network computing, IEEE Trans. Comput. 51 (9) (2002).
[45] Y. Gao, H. Rong, J.Z. Huang, Adaptive grid job scheduling with genetic algorithms, Future Gener. Comput. Syst. 21 (2005) 151–161.
[46] A. Litke, K. Tserpes, T. Varvarigou, Computational workload prediction for Grid oriented industrial applications: The case of 3D-image rendering, in: Proceedings of Cluster Computing and Grid, CCGrid2005, vol. 2, 2005, pp. 962–969.
[47] D. Pisinger, Algorithms for knapsack problems, Ph.D. Thesis, Dept. of Computer Science, University of Copenhagen, February 1995.
[48] A. Litke, K. Tserpes, K. Dolkas, T. Varvarigou, A task replication and fair resource management scheme for fault tolerant grids, in: Advances in Grid Computing—EGC 2005, in: Lecture Notes in Computer Science, vol. 3470, 2005, pp. 1022–1031.
[49] S. Martello, D. Pisinger, P. Toth, New trends in exact algorithms for the 0-1 knapsack problem, Eur. J. Oper. Res. 123 (2000) 325–332.

[50] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, 1990.
[51] D. Ghosh, Heuristics for knapsack problems: Comparative survey and sensitivity analysis, Fellowship Dissertation, IIM Calcutta, India, 1997.
[52] G. Ghosh, B. Goldengorin, The binary knapsack problem: Solutions with guaranteed quality, Research Report 01A64, University of Groningen, 2001.
[53] P.C. Chu, J.E. Beasley, A genetic algorithm for the multidimensional knapsack problem, J. Heuristics 4 (1) (1998) 63–86.
[54] C. Cotta, J.M. Troya, A hybrid genetic algorithm for the 0-1 multiple knapsack problem, in: G.D. Smith, N.C. Steele, R.F. Albrecht (Eds.), Artificial Neural Nets and Genetic Algorithms 3, Springer-Verlag, 1998, pp. 251–255.
[55] A.B. Simões, E. Costa, An evolutionary approach to the zero/one Knapsack problem: Testing ideas from biology, in: Proc. of the Fifth Int. Conf. on Neural Networks and Genetic Algorithms, ICANNGA'01, Prague, April 2001, pp. 236–239.
[56] S. Sahni, Data Structures, Algorithms, and Applications in Java, 2nd ed., Silicon Press, 2004.
[57] B.R. Preiss, Data Structures and Algorithms with Object-Oriented Design Patterns in C++, John Wiley & Sons, 1999.
[58] The Open Grid Services Architecture, Version 1.0, http://www.ggf.org/documents/GWD-I-E/GFD-I.030.pdf.
[59] OASIS Web Services Resource Framework (WSRF), http://www.oasis-open.org.
[60] Simple Object Access Protocol, http://www.w3.org/TR/soap/.

Antonios Litke received a diploma from the Computer Engineering and Informatics Department, University of Patras, Greece in 1999. Currently, he is pursuing his Ph.D. in the Telecommunication Laboratory of the Electrical and Computer Engineering Department of the National Technical University of Athens (NTUA), and works as a research associate in the Institute of Communication and Computer Systems, participating in numerous European Union (EU) and nationally funded projects. His research interests include Grid computing, resource management in heterogeneous systems, web services and information engineering.

Dimitrios Skoutas received his diploma from the Department of Electrical and Computer Engineering of the National Technical University of Athens in 2003. He is currently a Ph.D. candidate in the same department, where he also works as a software engineer and research associate. His research interests involve semantic web technologies, information retrieval and extraction from web sources, data mining, web services and grid computing.

Konstantinos Tserpes is a Research Associate in the Telecommunication Laboratory of the Institute of Communication and Computer Systems (ICCS), Athens, Greece. He graduated from the Computer Engineering and Informatics Department, University of Patras, Greece. In 2006 he received his MBA in Information Systems Management. He is currently undertaking his Ph.D. in the area of Grid Computing in the School of Electrical and Computer Engineering of the National Technical University of Athens. He has been involved in several EU and nationally funded projects, and his research interests revolve around Grid computing and its application and business extensions.

Theodora A. Varvarigou received a B. Tech degree from the National Technical University of Athens, Greece in 1988, MS degrees in Electrical Engineering (1989) and in Computer Science (1991) from Stanford University, California, USA in 1989, and a Ph.D. degree, also from Stanford University, in 1991. She worked at AT&T Bell Labs, Holmdel, New Jersey, USA between 1991 and 1995. Between 1995 and 1997 she worked as an Assistant Professor at the Technical University of Crete, Chania, Greece. Since 1997 she has been working as an Associate Professor at the National Technical University of Athens. Her research interests include Grid technologies, parallel algorithms and architectures, fault-tolerant computation, optimization algorithms and content management.