

## CHAPTER 1

---

# GAME-BASED MODELS OF GRID USERS' DECISIONS IN SECURITY AWARE SCHEDULING

---

JOANNA KOŁODZIEJ<sup>1</sup>, SAMEE ULLAH KHAN<sup>2</sup>, LIZHE WANG<sup>3</sup>, AND DAN CHEN<sup>4</sup>

<sup>1</sup>Cracow University of Technology, Institute of Computer Science, Cracow, Poland  
Email: jkolodziej [at] uck . pw . edu . pl

<sup>2</sup>North Dakota State University, Fargo USA,  
Email: samee . khan [at] ndsu . edu

<sup>3</sup>Center for Earth Observation, Chinese Academy of Sciences, China,  
Email: LZWang [at] ceode . ac . cn

<sup>4</sup>China University of Geosciences, Wuhan, China,  
Email: Danj43 [at] gmail . com

**Abstract:** This chapter summarizes our recent research on game-theoretical models of grid users' behavior in security aware scheduling. The main scheduling attributes, generic model of security aware management in local grid clusters and several scenarios of users' games are presented. Four GA-based hybrid schedulers have been implemented for the approximation of the equilibrium states of exemplary simple symmetric game of the grid end users. The proposed hybrid resolution meth-

ods are empirically evaluated through the grid simulator under the heterogeneity, security, large-scale and dynamics conditions.

### 1.1 INTRODUCTION

The concept of modern grid environment has grown far beyond the original Foster's grid idea of simple extension of conventional power networks. Today's grids are not just restricted to the physical infrastructure managed and utilized by the highly distributed users. Future generation grid environments are composed of various services and physical clusters in order to proceed the large amounts of data and information in an intelligent way. The management techniques in such systems should be able to group, predict, and classify different sets of rules, configuration directives, and environmental conditions. This management model must effectively deal with uncertainties in system information, that may be incomplete, imprecise or fragmentary.

The key scheduling objectives in today's computational grids, such as the maximization of the resource utilization and profits of the resource owners [9], may conflict with the users' requirements on security awareness in scheduling and system reliability at different levels of the grid global architecture [6]. Various system infections and external attacks may lead to unpredictable failures of the machines during the task executions. On the other hand, grid end users may define their own individual preferences and security condition for protection and personalization of their data and applications processed by the system. An effective grid scheduler must be then security-driven and resilient in response to all scheduling and risky conditions. It means that to achieve the successful tasks executions according the specified users' requirements, the relation between the assurance of secure computing services and the behavior of a resource node must be defined and analyzed [3]. The scheduler must be able to analyze and realize the users' decisions.

In many decision-making problems, most of the information is provided by humans, which is inherently non-numeric. Game-theoretical models, besides fuzzy logic methods [26], [27], [14], are the most promising technologies for illustrating and analyzing numerous users' decisions and behavior in highly parametrized heterogeneous environments [13]. In grid scheduling, game theory supports the analysis of the various users' strategies in multi-criteria resource allocation process. All scheduling criteria can be aggregated and defined as cumulative users' cost or payoff functions. Game-based models combined with the economic theory can capture many realistic scenarios in computational markets, computational auctions, grid and P2P systems.

This chapter summarizes the recent work of the authors on the game-theoretic models of the grid users behavior in security aware scheduling [5], [15], [19], [17], [20]. The scheduling process may be realized in two alternate scenarios, namely *risky* and *secure* modes. In the former, security conditions are ignored by the users, while in the later, the users allocate their tasks to available machines assuring task security demands. The specification of the security demand can be realized in two different

ways: (a) tasks can have security demands on resources to be allocated at and (b) resources can have security demands on tasks to be assigned to them. In this work we focus on the condition (a) of security requirements. The users' cost functions in the game are interpreted as the total costs of the secure execution of their tasks and the costs of the utilization of resources. The cumulative game cost function can be implemented as 2-step hierarchical optimization procedure and can be effectively minimized by the conventional global optimization methods, as well as by using the metaheuristics, that is recommended in large-scale dynamic systems such as grids and clouds.

The effectiveness of the GA-based grid schedulers in solving the security aware games of the grid users is illustrated in the case study provided in Sec. 1.4. We simulated simple symmetric game of the end users and minimized their cost functions by using four genetic schedulers working in risky and secure scenarios. All considered instances of grid system and game scenarios have been simulated by using the *Sim-G-Batch* grid simulator.

## 1.2 SECURITY AWARE SCHEDULING PROBLEMS IN COMPUTATIONAL GRIDS

Today's modern grid environment can be viewed as a complex highly parametrized distributed computing system that combines the infrastructure of intelligent computational platform and numerous services for the users and system administrators. Solving the scheduling problems in such an environment can be a big challenge for researchers and engineers, mainly due to the sheer size of the grid and its complex nature. Different types of scheduling problems in grids can be defined with respect to various grid characteristics and the requirements of the system administrators and system end users. All information generated by the numerous system components and received from all system "actors" must be "embedded" into the scheduling mechanism in order to achieve the desired performance of the grid [1], [34], [18].

Four main scheduling attributes are usually setup to specify the particular grid scheduling problem, namely: (a) the environment, (b) grid architecture, (c) task processing policy, and (d) tasks' interrelations. The type of grid environment can be static, where the number of the submitted applications and the available resources remain constant in a considered time interval, or dynamic, where states of the tasks and resources in the system may change over time.

The resource management and scheduling is usually organized in a hierarchical mode, that is a compromise among centralized and decentralized scenarios. In *centralized model*, there is a central authority, who has a full knowledge of the system. The primary disadvantages of this model is its limited scalability, lack of fault tolerance, and the difficulty in accommodating multiple local policies imposed by the resource owners. In *decentralized model*, local schedulers interact with each other to manage the tasks pool. In this model, there is no central authority responsible for resource allocation. In *hierarchical model*, few central meta-schedulers (or meta-brokers) interact with local job dispatchers in order to define the optimal schedules.

The local schedulers have knowledge about resource clusters, but they cannot monitor the whole system. The high level schedulers can communicate and exchange data and information by using an additional network infrastructure (usually Internet).

A specification of the tasks' processing policy is important in the identification of the particular scheduling problem. In the instantaneous mode the tasks are scheduled as soon as they are entered into the system. In batch scheduling, the submitted tasks are grouped into batches and the scheduler assigns each batch to the resources. Additionally, there can be some internal relations among submitted tasks, or each task can be processed independently to another.

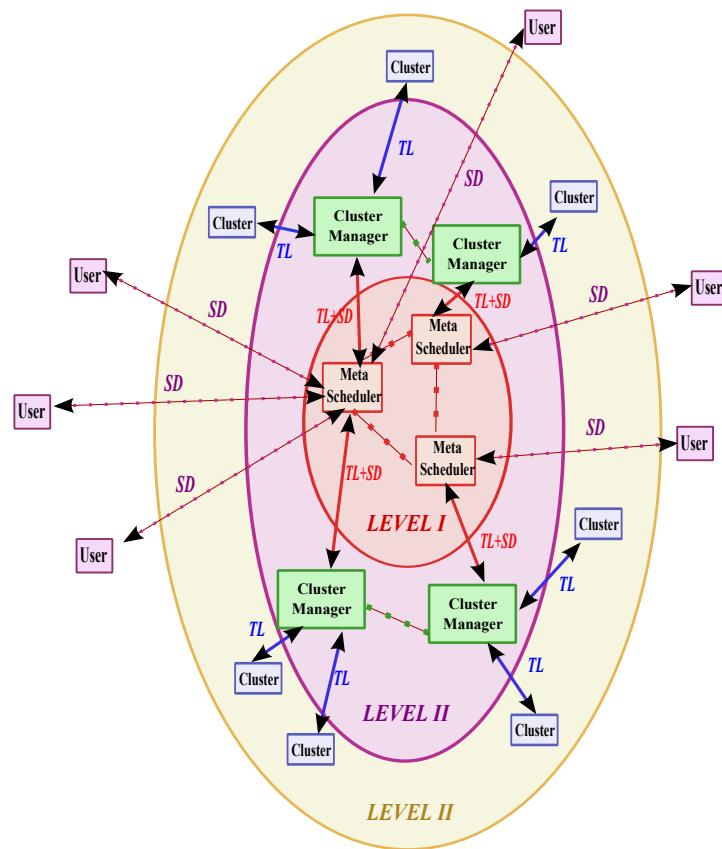
### 1.2.1 Generic Model of Secure Grid Clusters

A general security-aware grid model is based on the conventional hierarchical multi-level grid architecture [22]. However, the role of particular meta-scheduler is different when security is considered as additional criterion in the scheduling process. The meta-scheduler must analyze the security requirements for the execution of tasks and requests of the CG users for trustful resources available within the system. The local system brokers analyze "reputation" indexes of the machines received from the resource managers, send their suggestions to the meta-schedulers and control the resource allocation and communication between CG users and resource owners (or providers).

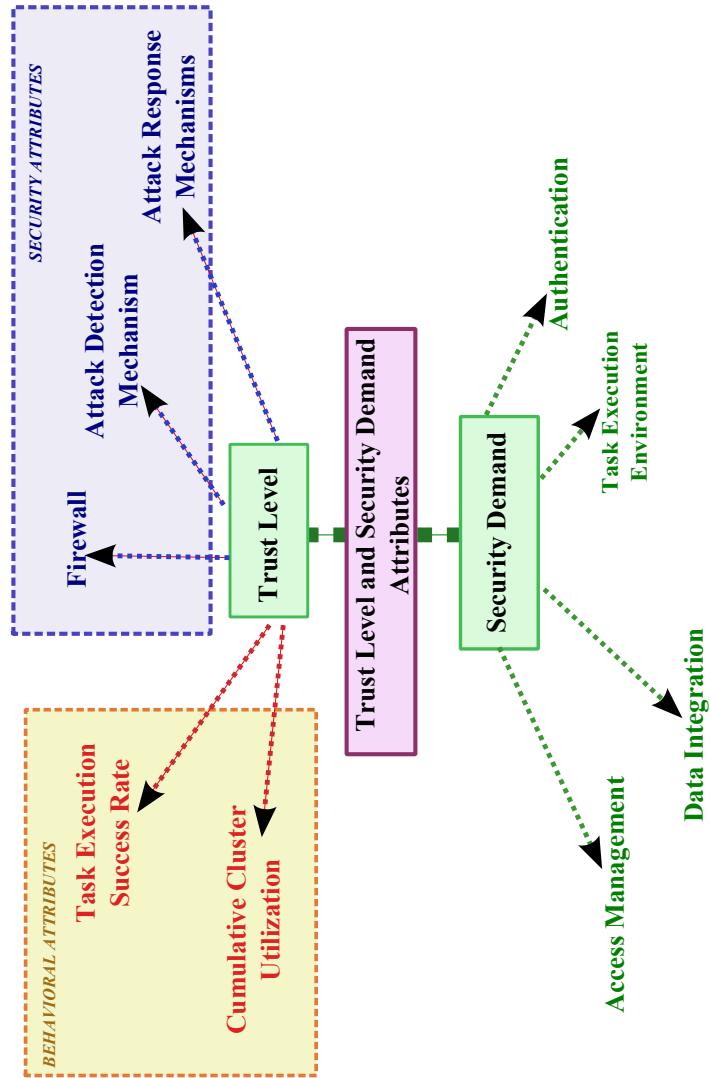
Fig. 1.1 depicts the 3-level architecture of a single security-aware grid cluster. The communication (and management) of different clusters can be realized at both meta-scheduler and local secure cluster manager levels.

The trust level and security demand parameters are generated by aggregation of several scheduling and system attributes as it is presented in Fig. 1.2. These attributes can be divided into two major classes: (1) behavioral and security attributes needed for the specification of trust levels of the grid resources, and (2) attributes necessary for specification of the security demand of the grid applications (see also [30]).

Song et al. in [29] have developed a fuzzy-logic trust model, where the major behavioral and intrinsic security attributes are represented as single scalar parameters. The task security demand in this model is supplied by the user's applications as request for authentication, data encryption, access control, etc. The trust level parameters of the resources are aggregated through a 2-level hierarchical fuzzy-logic based trust procedure. Song et al. have used their model in online scheduling, however it can be easily adapted also to the other types of grid scheduling problems. In independent batch scheduling it is used for the specification of new characteristics of tasks and resources in the grid system, namely security demand and trust level vectors. The *security demand vector*, denoted by  $SD = [sd_1, \dots, sd_n]$ , where  $sd_j$  denotes the security demand parameter of task  $j$ . The *trust level vector*, denoted by  $TL = [tl_1, \dots, tl_m]$ , is defined as a vector of trust level parameters  $tl_i$  for all resources in the system. The trust level parameters specify how much a grid user can trust the resource manager. The local cluster managers maintain the status of machines in their clusters and monitor the execution of the tasks assigned to those machines. The values of the  $sd_j$  and  $tl_i$  parameters are real fractions within the range [0,1] with 0



**Figure 1.1** The model of secure grid cluster



**Figure 1.2** The major attributes affecting the trust level and security demand in grid systems

representing the lowest and 1 the highest security requirements for a task execution and the most risky and fully trusted machine, respectively. A task can be successfully completed at a resource when a *security assurance condition* is satisfied. That is to say that  $sd_j \leq tl_i$  for a given  $(j, i)$  task-machine pair.

The probability of failure of a given machine ( $i$ ) during the execution of a given task ( $j$ ) is denoted by  $P_f[j][i]$  and can be defined by the negative exponential distribution function in the following way:

$$Pr_f[j][i] = \begin{cases} 0 & , \quad sd_j \leq tl_i \\ 1 - e^{-\alpha(sd_j - tl_i)} & , \quad sd_j > tl_i \end{cases} \quad (1.1)$$

where  $\alpha$  is interpreted as a failure coefficient and is a global parameter of the model.

The matrix of failure probabilities calculated for all possible task-machine pairs  $(j, i)$  is called as a *Machine Failure Probability* matrix and is denoted by  $Pr_f$ .

The process of matching  $sd_j$  with  $tl_i$  is similar to that of a real-life scenario where users of some portals, such as Yahoo!, are required to specify the security level of the login session.

### 1.2.2 Security Criterion in Grid Scheduling

Various scheduling problems specified by different combinations of the scheduling attributes can be in fact modelled by using one general methodology. To our best knowledge, there is no commonly used notation for the classification of the grid scheduling problems and conventional scheduling notation of Graham et al. [11] is insufficient for the dynamic scheduling in large-scale distributed environments. In this chapter we focus on the fundamental type of the scheduling problem in computational grids, namely independent batch scheduling. where it is assumed that tasks are clustered and grouped into batches and there is no inter-relations among them. The trust level and security demand parameters introduced in the previous section are used for the specification of tasks and machines in the system, that can be characterized by the following structures:

(a) **Tasks:**

- set of tasks' labels  $N = \{1, \dots, n\}$ , ( $n$  – number of tasks in the batch);
- workload vector  $WL = [wl_1, \dots, wl_n]$ , where  $wl_j$  – workload parameter of task  $j$  expressed in Millions of Instructions (MI);
- security demand vector –  $SD = [sd_1, \dots, sd_n]$ ;

(b) **Machines:**

- set of machine labels –  $M = \{1, \dots, m\}$ , ( $m$  – total number of machines in the batch);
- computing capacity vector –  $CC = [cc_1, \dots, cc_m]$ , where  $cc_i$  – computing capacity parameter of machine  $i$  expressed in Millions of Instructions Per Second (MIPS);

- *ready times vector* –  $ready\_times = [ready_1, \dots, ready_m]$ , where  $ready_i$  is a ready time of machine  $i$ , which expresses the time needed for reloading this machine after finishing the last assigned task;
- *trust level vector* –  $TL = [tl_1, \dots, tl_m]$ .

Tasks in this model may be considered as monolithic applications or meta-task with no dependencies among the components. The workloads of tasks are usually estimated by using some system prediction methodologies [12], historical data or users' specification. The term "machine" in this work is related to a single or multiprocessor computing unit or even to a local small-area network.

For each pair  $(j, i)$  of task-machine labels, the coordinates of  $WL$  and  $CC$  vectors can be used for an approximation of the completion time of the task  $j$  on machine  $i$ . This completion time is denoted by  $ETC[j][i]$  and can be calculated in the following way:

$$ETC[j][i] = \frac{wl_j}{cc_i}. \quad (1.2)$$

All  $ETC[j][i]$  parameters are defined as the elements of an  $ETC$  matrix,  $ETC = [ETC[j][i]]_{n \times m}$  in ETC matrix model [2]. The elements in the rows of the ETC matrix define the estimated completion times of a given task on particular machines, and elements in the column of the matrix are interpreted as approximate times of the completion of particular tasks on a given machine.

The ETC matrix model can be used for the formal implementation of the main scheduling criteria, namely *Makespan* and Flowtime. Let us first introduce two representations of the schedules in the system. Let  $\mathcal{S}_\infty$  denotes the set of all permutations **with repetition** of the length  $n$  over the set of machine labels  $M$ . An element  $Sch(1) \in \mathcal{S}_\infty$  is termed a *schedule* and it is encoded by the following vector:

$$Sch(1) = [i_1, \dots, i_n]^T, \quad (1.3)$$

where  $i_j \in M$  denotes the number of the machine on which the task labeled by  $j$  is executed.

This encoding method is called *direct representation* of the schedule and can be easily transformed into into *permutation-based representation*, that is defined as a vector  $Sch(2)$  of labels of tasks assigned to the machines. For each machine the labels of the tasks assigned to this machine are sorted in ascending order by the completion times of the tasks. Formally, this kind of schedule encoding method can be defined in the following way.

Let us denote by  $\mathcal{S}_2$  the set of all permutations **without repetitions** of the length  $n$  over the set of task labels  $N$ . A permutation  $Sch \in \mathcal{S}_2$  is called a *permutation-based representation* of a schedule in CG and can be defined by the following vector:

$$Sch(2) = [Sch_1, \dots, Sch_n]^T, \quad (1.4)$$

where  $Sch_i \in N$ ,  $i = 1, \dots, n$ .

In this representation some additional information about the numbers of tasks assigned to each machine is required. The total total number of tasks assigned to a machine  $i$  is denoted by  $\widehat{Sch}_i$  and is interpreted as the  $i$ -th coordinate of an assignment vector  $\widehat{Sch}(2) = [\widehat{Sch}_1, \dots, \widehat{Sch}_m]^T$ , which defines in fact the loads of grid machines.

The *Makespan* and *Flowtime* are considered in this chapter as the main scheduling criteria and may be expressed in terms of the completion times of the machines by using the ETC matrix model. A *completion time*  $C[i]$  of the machine  $i$  is defined as the sum of the ready time parameters for this machine and a cumulative execution time of all tasks actually assigned to this machine, that is to say:

$$C[i] = ready_i + \sum_{j \in Task(i)} ETC[j][i], \quad (1.5)$$

where  $Task(i)$  is the set of tasks assigned to the machine  $i$ .

The  $C_i$  parameters are the coordinates of the following completion vector:

$$C = [C[1], \dots, C[m]]^T \quad (1.6)$$

Vector  $C$  is used for calculating the *Makespan*  $C_{max}$  in the following way:

$$C_{max} = \max_{i \in M} C[i]. \quad (1.7)$$

In terms of ETC matrix model, a *Flowtime* for a machine  $i$  can be calculated as a workflow of the sequence of tasks on a given machine  $i$ , that is to say:

$$F[i] = ready_i + \sum_{j \in Sorted(i)} ETC[j][i] \quad (1.8)$$

where  $Sorted(i)$  is the set of tasks assigned to the machine  $i$  sorted in ascending order by the corresponding *ETC* values.

The cumulative *Flowtime* in the whole system is defined as the sum of  $F[i]$  parameters, that is:

$$F = \sum_{i \in M} F[i] \quad (1.9)$$

In security aware scheduling, the grid cluster or the grid resource may be not accessible to the global meta-scheduler when being infected with intrusions or by malicious attacks. The scheduler may analyze the machine failure probability matrix  $P_{f_r}$  in order to minimize the failure probabilities for task-machine pairs, or he can perform an “ordinary” scheduling without any preliminary analysis of the security conditions. The scheduler’s strategies give rise to two modes of processing the grid applications, namely *secure* and *risky* modes.

In secure scenario all security and resource reliability conditions are verified for all possible task-machine pairs before the scheduling process is initialized. Each meta-scheduler try to assign the tasks to the machines with possible minimal probabilities of failures of the machines during the executions of these tasks. There is some additional “cost” of the verification of security assurance condition for a given task-machine pair, that can be defined as the product of the probability  $P_{f_r}[j][i]$  and the

time  $ETC[j][i]$ . In this case the completion time of the machine  $i$  is denoted by  $C^s[i]$  and can be calculated as follows:

$$C^s[i] = ready_i + \sum_{\{j \in Tasks(i)\}} (1 + Pr_f[j][i]) ETC[j][i] \quad (1.10)$$

where  $Tasks(i)$  denotes the set of tasks assigned to the machine  $i$  in a given batch.

In this mode the *Makespan* and *Flowtime* can be expressed as follows:

$$C_{max}^s = \max_{i \in M} C^s[i]. \quad (1.11)$$

$$F^s = \sum_{i \in M} F^s[i] \quad (1.12)$$

where

$$F^s[i] = ready_i + \sum_{j \in Sorted[i]} (1 + Pr_f[j][i]) ETC[j][i] \quad (1.13)$$

and  $Sorted[i]$  denotes the set tasks assigned to the machine  $i$  sorted in ascending order by the corresponding  $ETC$  values.

**Risky Mode.** In this scenario all secure and failing conditions are ignored by the meta schedulers. The tasks are assigned to the machines based on the analysis of the  $ETC$  matrix. The estimated completion times of machines are calculated as in Eq. (1.5). If failures of machines are observed, then the unfinished tasks are temporarily moved into the backlog set, and then re-scheduled in the way as in the secure mode. The total completion time of machine  $i$  ( $i \in M$ ) in this case can be defined as follows:

$$C^r[i] = C[i] + C_{res}^s[i] \quad (1.14)$$

where  $C[i]$  is calculated by using the Eq. (1.5) for tasks primarily assigned to the machine  $i$ , and  $C_{res}^s[i]$  is the completion time of machine  $i$  calculated by using the Eq. (1.10) for rescheduled tasks, i.e. the tasks re-assigned to the machine  $i$  from the other resources.

The formulas for *Makespan* and *Flowtime* in this mode are defined in the following way:

$$C_{max}^r = \max_{i \in M} C^r[i]. \quad (1.15)$$

$$F^r = \sum_{i \in M} F^r[i] \quad (1.16)$$

where

$$F^r[i] = ready_i + \sum_{j \in Sorted[i]} ETC[j][i] + \sum_{j \in Sorted_{res}[i]} (1 + Pr_f[j][i]) ETC[j][i] \quad (1.17)$$

and  $Sorted_{res}[i]$  denotes the set of *rescheduled* tasks assigned to the machine  $i$  sorted in ascending order by the corresponding  $ETC$  values

### 1.2.3 The Requirements of Grid End Users for Scheduling

The complexity of the scheduling scenarios and the roles of the system administrators, service providers, and grid power users strongly depend on the end users Quality of Service (QoS) requirements and specific scheduling and task execution criteria, such as requirements for specifying their tasks as monolithic, atomic applications or meta-tasks, access to the remote data or detailed characteristics of the system resources. The users would like to know the “reputation” indexes of the grid clusters. These parameters can be specified by the local cluster managers or resource owners as trust level parameters (see Sec. 1.2.1). The user may wish to target particular types of resources (e.g. SMP machines), but should not be concerned with the type of resource management on the grid, nor with the resource management systems on individual resources on the grid.

The trust indexes can be used in the specification of *reliability of grid resources*. In some cases, the machines can fail during the task execution or may be unavailable for some users or some types of applications due to restrictions and special access policies defined by the owners. The information about the current state and reliability of grid resources is needed for an effective management of the users tasks and for the reduction of costs of possible resource failures. In the case of resource failure the system administrators can activate re-scheduling or task migration procedures, and preemption policies. On the other hand, the end users may specify some special requirements on the assignment of their tasks to the most trustful resources, no matter how expensive this assignment is. Based on the values of trust indexes, the user should be able to estimate the security demands for his tasks on the available resources. In today’s grid the users should be able to monitor the current status of their tasks and applications. The secure access to the system information, internal services and data requires the specification of some standardized authentication and authorization mechanisms. These standardized authentication certificates can be digitally signed by a certificate authority, and kept in the user’s repository, that is recognized by the resources and resource owners. All such certificates can be automatically created by the user’s interface application during the submission of his tasks.

Trying to satisfy the QoS requirements of the end users, the grid managers can work in one of the following scenarios:

- **Cooperativeness:** In this case the system “actors” can form a coalition to plan in advance their actions;
- **Non-cooperativeness:** In this scenario all system users and managers act independently of one another;
- **Semi-cooperativeness:** In this model each user can select a partner for the cooperation.

The above mentioned relations can also give us a brief characteristics of the end users behavior and decisions. In most of the scheduling scenarios, it is assumed that the end users submit their tasks independently. However, in some cases the local coalition of the end users can improve the effectiveness of the system, especially in

the cases of considering security to be an extra scheduling criterion embedded into the scheduling model. The strategies and decisions of the grid users and managers of all types can be modelled by using the game theory.

### 1.3 GAME MODELS IN SECURITY AWARE GRID SCHEDULING

Modelling of the autonomous decisions of the grid users and personalization of the users' data and requirements are one of the most important issues in today's grid scheduling. This modelling may be difficult due to the different access policies and sometimes conflicting requirements of the end users. Game-based models, that can be additionally supported sometimes by the economic strategies, can capture many realistic scheduling scenarios and allows to aggregate all scheduling criteria into the cumulative users' cost or pay-off functions, which makes such models very useful in the analysis of the various users' strategies in the resource allocation process.

A short description of the relations of grid managers and administrators provided in the previous section, may be used also for the characteristics of the games of the grid users of all types, not just the end users. Therefore, we may define three basic types of grid user games, namely *non-cooperative*, *cooperative* and *semi-cooperative* games [4], in the following way:

- **non-cooperative game:** in this game the players take autonomous decisions, independently one to another. In fact, in the realistic grid environments the end users usually are unable to cooperate due to the sheer size of the whole infrastructure and sometimes conflict of their interests. Also the resource owners and providers try to increase their own profits and keep their resources busy for a long time.
- **cooperative game:** in this scenario the players can form the coalition and make the collective decisions on the assignments of their tasks. The whole coalition can specify its own strategies. This game model can be very useful for illustrating the negotiations of the meta-schedulers from different clusters and local cluster managers.
- **semi-cooperative game:** in this case each user can choose (randomly) another user (player) for cooperation. This game is usually proposed as a multi-round auction to incorporate the task rescheduling in the case of failures of machines.

Each grid user at various system levels can have different privileges and access to the resources. Therefore, two following scenarios can be additionally consider in each of the above mentioned games:

- **Symmetric scenario.** In this case there are no special privileges in the resource usage for the grid users.
- **Asymmetric scenario.** In this case there is a privileged user (Leader), who can have full access to resources as opposed to the rest of users who can be

granted only limited access to resources. The tasks submitted by the Leader are scheduled first.

In the following subsections we provide a brief characteristics of the above mentioned scenarios in the context of both end users' and system managers' games.

### 1.3.1 Symmetric and Asymmetric Noncooperative Games of the End Users

One of the main benefits of the game-based scheduling and resource management in CGs is that it enables a scalability and personalization of the decision-making processes of grid end users. Due to the sheer scale of grid systems, the non-cooperative game is a potential model for integrating security and resource reliability requirements in grid scheduling. This section presents two different general scenarios of the non-cooperative grid users behaviors, namely *symmetric* and *asymmetric* strategic game models.

**1.3.1.1 Non-Cooperative Symmetric Game** In *symmetric* non-cooperative users' game the users are selfish and do not make any collective decisions. Also there are no priorities and privileges in their access to the resources. An illustrative example of the symmetric game can be a scheduling scenario in which each player submits approximately equal amount of tasks. This game can be defined by the following parameters:

$$G_A = (A; \{X_a\}_{a=1,\dots,A}; \{Q_a\}_{a=1,\dots,A}), \text{ where:}$$

- $A$  is the number of grid users;
- $\{X_1, \dots, X_A\}$ ; are the sets of users' strategies;
- $\{Q_1, \dots, Q_A\}; Q_a : X_1 \times \dots \times X_A \rightarrow \mathbb{R}; \forall_{a=1,\dots,A}$  is the set of users' cost functions.

The elements of sets  $X_a$  ( $a = 1, \dots, A$ ) are defined as users strategy vectors and are denoted by  $Pl_1, \dots, Pl_A$ . Function  $Q_a$  estimates the cumulative cost of scheduling all tasks submitted by user  $a$ . The players try to minimize simultaneously their cost functions  $Q_a$  in the game.

An optimal state of the whole game is called **an equilibrium state (point)** and it is represented by the multi-vector  $(\widehat{Pl}_1, \dots, \widehat{Pl}_A)$  of the users' strategies, for which the following condition holds:

$$\begin{aligned} Q_a(\widehat{Pl}_1, \dots, \widehat{Pl}_A) &= \\ &= \min_{Pl_a \in X_a} Q_a(\widehat{Pl}_1, \dots, \widehat{Pl}_{(a-1)}, Pl_a, \widehat{Pl}_{(a+1)}, \dots, \widehat{Pl}_A) \end{aligned} \quad (1.18)$$

for all  $a = 1, \dots, A$ .

The equilibrium point can be interpreted as a steady state of the a strategic game, in which each player holds correct expectations concerning the other players behavior<sup>1</sup>.

<sup>1</sup>In the case of continuous players' cost functions the equilibrium state of the game is called the *Nash equilibrium* [7].

It can be observed from Eq. (1.18) that the problem of generation of the equilibrium points is a multi-objective global optimization task, and the equilibrium states of the game should be Pareto-optimal [31, 25]. Usually, in the complex game scenarios it is very hard to verify this Pareto-optimality. In this chapter we focus on the non-zero sum games, where the strategies of the players are not opposite, i.e. the sum or the values of all players cost functions  $Q_a$  is not 0. In such a case, the equilibrium points are the results of minimization of a *multi-cost game function*  $Q$  defined as follows.

Let us denote by  $\min Q_a, (a = 1, \dots, A)$ , the minimal value of the function  $Q_a$  calculated for each user  $a$ , that is to say:

$$\min Q_a = \min_{Pl_a \in X_a} \{Q_a(Pl_1, \dots, Pl_A)\}. \quad (1.19)$$

The results of the global minimization of the following *game multi-cost* function  $Q : X_1 \times \dots \times X_A \rightarrow \mathbb{R}$ :

$$Q(Pl_1, \dots, Pl_A) = \sum_{a=1}^A \frac{1}{A} (Q_a(Pl_1, \dots, Pl_A) - \min Q_a), \quad (1.20)$$

is an equilibrium state of non-cooperative non-zero sum symmetric game of the grid users, which satisfies the condition of the Pareto-optimality [25]. It should be noted, that the function  $Q$  is a special case of the weighed or weighed distance  $L^p$  metric function with  $p = 1$  [32]. The values of all  $Q_a$  functions are non-negative, and the weight coordinates are strictly positive, which means that the global solutions of the problem defined in Eq. (1.20) are Pareto-optimal.

**1.3.1.2 Asymmetric Scenario – Stackelberg Game** Although the symmetric game is very easy for the implementation, it can be difficult sometimes to realize this scenario in practical applications. Due to the cross-domain access, authorization and resource management features of the grid system, the grid users have different access policies to the resources and their relations are rather asymmetric with regard to resource usage privileges. Also the amount of particular tasks submitted by the users may be different.

A Stackelberg game is one of the simplest and well studied model for illustrating the asymmetric scenario of the behavior of non-cooperative grid users. In this game one user acts as a *Leader*, and the rest of players (users) are his *Followers*.

The Stackelberg games have been well-studied in the game theory literature (see, e.g. [4]). Roughgarden [28] used the Stackelberg game model for scheduling in peer-to-peer networks of machines with load-dependent latencies. The total latency of the system has been minimized in that game.

The following examples illustrate some real-life grid scenarios, where the Stackelberg game model can be applied (see also [20]):

- One user (Leader) may have the full access to resources, while the other users – just limited access.
- Some tasks can have critical deadlines (especially in online scheduling) and they can be sent by the Leader to schedule them first.

- One user (Leader) can be the owner of a large portion of the tasks in the batch.
- The Leader can have some special requirements on the “reputation” of the available resources. He may wish to schedule his tasks to the most trustful resources (secure machines).
- The high heterogeneity of tasks in grid usually has a great impact on the system performance. Therefore, the Leader could create a small batch of the most time consuming tasks as the backlog set of grid applications, in order to “balance” the computational loads of machines during the scheduling.

Formally, the Stackelberg game of the grid users can be defined as two-level procedure as follows [20]:

- **Leader’s action I:** the game is initialized by the Leader, his initial strategy is denoted by  $\widetilde{Pl}_1 = [\widetilde{j}_1, \dots, \widetilde{j}_{k_1}]$ , where  $k_1$  is the number of tasks submitted by the Leader.
- **Followers’ action:** Followers minimize simultaneously their cost functions relative to the Leader’s action I:

$$\begin{cases} Pl_2^{Fol} = \arg \min_{(Pl_2 \in X_2)} \{Q_2(\widetilde{Pl}_1, Pl_2, \dots, Pl_A)\} \\ \vdots \\ Pl_A^{Fol} = \arg \min_{(Pl_A \in X_A)} \{Q_A(\widetilde{Pl}_1, \dots, Pl_A)\} \end{cases} \quad (1.21)$$

- **Leader’s action II:** Leader updates his strategy by minimizing his cost function  $Q_1$  (see also Eq. (1.30)) taking into account the result of Followers’ actions. The following vector  $Pl^G = [Pl_1^{Lead}, Pl_2^{Fol}, \dots, Pl_A^{Fol}]$ , where:

$$Pl_1^{Lead} = \arg \min_{(Pl_1 \in X_1)} Q_1(Pl_1, Pl_2^{Fol}, \dots, Pl_A^{Fol}) \quad (1.22)$$

is a solution of the whole game.

In the above formulas  $X_1$  denotes the set of the Leader’s strategies and  $Pl^{Fol} = [\widetilde{Pl}_1, Pl_2^{Fol}, \dots, Pl_A^{Fol}]$  denotes a *Followers’ Vector*, which is interpreted as the result of the Followers’ action. It should be noted that the Followers play an “ordinary” non-cooperative symmetric game, but they must know the Leader’s action. The game multi-cost function  $Q$  in this case can be defined in the following way:

$$Q_{Stac} = \frac{1}{A} Q_1 + Q_{Fol}; \quad (1.23)$$

where  $Q_1$  is the Leader’s cost function and

$$Q_{Fol} := \frac{A-1}{A} \sum_{a=2}^A Q_a; \quad (1.24)$$

is a *Followers' cost function*. An optimal solution of the whole game is called *Stackelberg Equilibrium*.

Non-cooperative games are commonly used for modelling the end users relations. However, those games can be also applied for illustrating the realistic relations of the resource owners. Indeed, in many cases the main aim of the machine owners is to achieve the high profits from renting their computational infrastructure and they usually send their offers to the local service and resource providers independently.

### 1.3.2 Cooperative and Semi-cooperative Game Scenarios

In practical applications, various levels of the cooperativeness of the grid users can be modelled by using the auction mechanism. In auctions there are two main groups of participants: sellers (resource owners) and buyers (grid end-users). The cooperation between users to form a coalition and win the auction is possible, but usually the users behave selfishly. The auction mechanism can be defined in many ways (e.g. English, Dutch, First and Second Price auctions). A task is awarded to the highest bidder, meaning that the user or system manager that is the best fit to execute the task wins. For the work done, each grid user is compensated by side-payments. All of which differ in terms of whether they are performed as open or closed auctions and the offer price for the highest bidder. The users' strategies in particular auctions are discussed e.g. in [10].

Khan et al. [16] has observed that in the case of conventional resource providing in the grid systems and data centers the auction mechanism is effective in the case of semi-cooperation or cooperation of the grid users. Especially in the case of immediate rejection of tasks, the semi-cooperative  $p$ -round sealed-bid auction, which incorporates task reallocation. The user who is unable to execute the task, on random, chooses another user and passes the un-executable task to it. If the newly chosen user can execute the task then the reallocating procedure finishes, otherwise, another user is selected on random. This process is repeated  $p - 1$  times, where  $p$  is the number of the users.

Another proposal of Khan is the cooperative scenario [15], where all users collectively negotiate and deliberate to come up with a task allocation that is beneficial to the system as a whole. in such a scenario, some individual users may not be content with the decision of the coalition, but the resulting allocation is efficient in the sense that it aims to reduce the *Makespan* and *Energy* consumed by the system, and provides a load balanced task allocation.

Both semi-cooperative and cooperative scenarios presented by an et al. may be adapted to modelling the behavior of meta-schedulers and local job dispatchers in hierarchical grid model presented in Fig. 1.1. In this model the end users can specify their requirements and the task and resource managers are responsible for seeking and selecting the best offer for their "clients" according to the security scheduling criteria.

### 1.3.3 Online Scheduling Games

The idea of the simple auction models used in batch scheduling has been extended by Kwok et al. [21] to the online scheduling. It is model the resource management and global scheduling policies are specified at the intra- and inter-site levels in the 3-levels hierarchical grid structure. In the intra-site bidding each machine owner in the site, who acts selfishly, declares the “execution capability” of the resource. The local manager monitors these amounts and sends a single value to the global scheduler. In the inter-site bidding the global scheduler should allocate tasks according to the values sent by the local dispatchers. Formally, the users decisions can be planned and executed at three different levels and therefore three different game theoretic task allocation and execution problems:

- *Intra-Site Task Execution Strategies:* This problem concerns about the strategies of the participating computers inside a local grid cluster (site). Specifically, although the various computers “participate” in the making up of the grid cluster, the owner of each individual machine acts selfish in that he only wants to execute tasks from local users but does not want to contribute to the execution of remote jobs.
- *Intra-Site Bidding:* This problem concerns about the determination of the advertised “execution capabilities” for tasks submitted to one global meta-scheduler. The meta-scheduler needs to know all the sites’ execution capabilities. The local job dispatcher can then “moderate” all declarations about the trust levels and computational capacities of the machines and send the offers to the global scheduler. For example, if the local job dispatcher is aggressive in task execution, it could use the “minimization” approach taking the minimum value of the declarations from all the machine owners. The best strategies for the resource owners can be selected by using the auction theory.
- *Inter-Site Bidding:* Similar to the intra-Site situation, at the inter-site level, the various local job dispatchers also need to formulate game theoretic strategies for computing the single representative value of the job execution time to be sent to the global scheduler.

Song et al. in [29] and [30] showed that different combinations of the above games result in different grid management structures. For a semi-selfish grid, the intra-site games are non-cooperative while the inter-site game is cooperative. This model fits most today’s grid and cloud environments, cause these infrastructures are usually initialized and built after some cooperative negotiations at the organization level. The results presented in [30] are extended by Wu et al. in [33]. The authors consider the heterogeneity of fault-tolerance mechanism in a security-assured grid job scheduling and define four types of GA-based online schedulers for the simulation of fault-tolerance mechanisms.

## 1.4 CASE STUDY: APPROXIMATING THE EQUILIBRIUM STATES OF THE END USERS SYMMETRIC GAME BY USING THE GENETIC METAHEURISTICS

The problem of solving the finite strategic game remains challenging especially in real-life approaches. The values of the game cost functions  $Q$  defined in Eqs. (1.20) and (1.23) can be computed, if the cost functions of all players have been first minimized. Therefore the problem of the minimization of  $Q$  function can be defined as a two-level hierarchical global optimization procedure, that requires some simple and fast optimization methodologies implemented at each level.

In this section we provide as a simple case study, a simulation of symmetric non-cooperative game of the grid end users. The cumulative time of the execution of tasks submitted to the system, the cost of the utilization of the resources and the cost of the verification of security condition (or the failures of the machines during the task executions) are interpreted as a cumulative cost of playing the game and are specified for each user as its game cost function. The  $Q$  function has been minimized by using four types of genetic-based risk-resilient metaheuristics.

### 1.4.1 Specification of the Game

**1.4.1.1 Characteristics of Game Players and Decision Variables** We assume that the general scenario of the game is identical with that defined in Sec. 1.3.1.1 and  $A$  denotes the number of grid users (players). The total number of tasks  $n \in N$  in a given batch can be expressed as the sum of tasks submitted by all users, i.e.

$$n = \sum_{a=1}^A k_a, \quad (1.25)$$

where  $k_a$  is the number of tasks of the user  $a = 1, \dots, A$ .

Each user  $a$  controls his strategic variables. These variables are expressed by the labels of the tasks submitted by a given user and defined as the following *user's strategy vector*:

$$Pl_a = [j_{(\widehat{k_{(a-1)}}+1)}, \dots, j_{(\widehat{k_{(a-1)}}+k_a)}] \quad (1.26)$$

where  $\widehat{k_{(a-1)}} = k_1 + \dots + k_{(a-1)}$

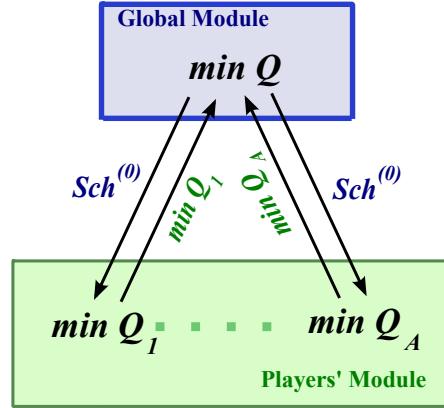
The schedules can be then expressed by the following vectors of the users' parameters:

$$Sch(1) = [i_1^1, \dots, i_{k_1}^1, \dots, i_{(\widehat{k_{(a-1)}}+1)}^a, \dots, i_{(\widehat{k_{(a-1)}}+k_a)}^a, \dots, i_{k_A}^A], \quad (1.27)$$

in the direct representation, and

$$Sch(2) = [Pl_1, \dots, Pl_A], \quad (1.28)$$

in permutation-based representation (see Sec. 1.2.2).



**Figure 1.3** Hierarchical procedure of solving non-cooperative symmetric game of grid users

**Solving the Grid Users Game.** The general idea of the optimization of the game cost function  $Q$  presented in Fig. 1.3. The optimization procedure is composed of two cooperating modules: **Global Module**, where the values of the function  $Q$  are calculated and optimized, and the **Players' Module** - where the users' cost functions  $Q_a$  are optimized.

The communication procedure between *Global* and *Players' Modules* can be defined as follows: Let us denote by  $Sch^{(0)}$  permutation-based representation of an initial schedule generated in the *Global Module*, i.e.  $Sch^{(0)} = [Pl_1^{(0)}, \dots, Pl_{Play}^{(0)}]$ , where  $Pl_a^{(0)}$  is the initial strategy vector of the user  $a$  (see Eq. (1.26)). Vector  $Sch^{(0)}$  is replicated and its copies are sent to the *Players' Module* - one copy per user. Then, each user independently optimizes his game cost function<sup>2</sup> by changing the allocations of just his own tasks. As the result of this minimization, the optimal values of the  $Q_a$  cost functions are calculated:

$$\left\{ \begin{array}{l} minQ_a^{(0)} = \min_{(Pl_i \in X_1)} Q_1(Pl_1, Pl_2^{(0)}, \dots, Pl_A^{(0)}) \\ \vdots \\ minQ_A^{(0)} = \min_{(Pl_A \in X_A)} Q_A(Pl_1^{(0)}, \dots, Pl_{A-1}^{(0)}, Pl_A) \end{array} \right. \quad (1.29)$$

These values are sent back to the **Global Module**, where the objective function for the whole game  $Q$  is calculated for the schedule  $Sch^{(0)}$ . Similar procedure can be defined for the direct representation of the schedules.

**1.4.1.2 Game Cost Functions** The cost functions  $Q_a$  ( $a = 1, \dots, A$ ) calculated for all users must, beyond the conventional scheduling criteria such as *Makespan*, *Flow-time* and the resource utilization [9], express the estimated costs of the verification

<sup>2</sup>Note that the users costs optimization in the **Players' Module** can be implemented as a parallel multi-threaded procedure, which can speed-up the whole process.

of the security conditions and possible machine failures. The users have to “pay” an additional “fee” for the secure allocation of their tasks in the machines. Therefore, functions  $Q_a, a \in \{1, \dots, A\}$  can be defined as weighed or simply direct sum of the following three components:

$$Q_a = Q_a^{(ex)} + Q_a^{(util)} + Q_a^{(sec)}, \quad (1.30)$$

where:

- $Q_a^{(ex)}$  indicates the user's task execution cost,
- $Q_a^{(util)}$  denotes the resource utilization cost, and
- $Q_a^{(sec)}$  is the cost of security-assured allocation of the user tasks .

Similarly as for conventional scheduling objectives in independent batch scheduling, ETC Matrix model is useful for the specification of all cost functions of the grid users.

**Task Execution Cost.** The cumulative cost of execution of the tasks submitted by particular users can be calculated as an average completion time of their tasks. In terms of the completion times of machines (see Eq. (1.5)) the task execution cost  $Q_a^{(ex)}$  for user  $a$  can be defined as follows:

$$Q_a^{(ex)} = \frac{\sum_{j=k_{a-1}+1}^{k_a} C[j][i]}{C_m \cdot k_a}, \quad (1.31)$$

where  $C[j][i]$  denotes the completion time of a task  $j$  on a given machine machine  $i$  and it is calculated in the following way:

$$C[j][i] = ETC[j][i] + ready[i], \quad (1.32)$$

and  $C_m$  is the maximal completion time of all tasks submitted by the user  $a$ .

**Resource Utilization Cost .** The grid user's utility function is usually defined as a cost of buying free CPU cycles [9]. In this work the utilization cost paid by the user  $a$  is calculated as amount of average idle time of machines on which his tasks are executed. This amount depends (is proportional) of the total time spent for execution of his tasks on those machines. is defined by This cost depends on the completion times of the user's tasks. The utility function  $Q_a^{(util)}$  is defined as follows:

$$Q_a^{(util)} = \sum_{i \in machines(a)} \left(1 - \frac{C_{(a)}[i]}{C_{max}}\right) \cdot Idle\_Factor[i] \quad (1.33)$$

where  $machines(a)$  denotes the set of machines, to which all tasks of the user  $a$  are assigned and  $C_{max}$  refers to the *Makespan*. The completion time of a given machine  $i \in machines(a)$ , denoted by  $C_{(a)}[i]$ , is calculated in the following way:

$$C_{(a)}[i] = ready[i] + \sum_{\substack{j \in N: \\ Sch_j(1)=i}} ETC[j][i] \quad (1.34)$$

where  $Sch_j(1)$  is the value of  $j$ -th coordinate in a given schedule vector  $Sch(1)$  (or  $Sch(2)$  – both implementations of the schedules may be used in Eq. (1.34). The following expression:

$$\left(1 - \frac{C_{(a)}[i]}{C_{max}}\right) \cdot Idle\_Factor[i], \quad (1.35)$$

in Eq. (1.33) is interpreted as an idle time of machine  $i$  calculated for a given user  $a$ . This is just a “portion” of the total idle time of machine  $i$ , and it is proportional to the time of execution of all tasks of the user  $a$  assigned to this machine. This proportion is specified by the coefficient  $Idle\_Factor[i]$  in the following way:

$$Idle\_Factor[i] = \frac{\sum_{j \in Tasks_{(a)}[i]} ETC[j][i]}{C_{(a)}[i]} \quad (1.36)$$

where  $Tasks_{(a)}[i]$  is the set of the tasks of the user  $a$  assigned to the machine  $i$ .

**Security-Assurance Cost.** The security-assurance cost of scheduling the tasks of the user  $a$ , denoted by  $Q_a^{(sec)}$  in Eq. (1.30), depends on the scheduling scenario (risky or secure) and the result of the verification of security condition by the local cluster manager.

In the **Risky mode** the “security” components of the functions  $Q_a$  are in fact not calculated, i.e.  $Q_a^{(sec)} = 0, \forall a = 1, \dots, A$ . However, some machines may fail during the tasks executions and rescheduling procedure of those tasks must be activated. Therefore the security cost in this case is calculated as follows:

$$Q_l^{(sec)}[ris] = \sum_{j \in Res(a)} \frac{P_f[j][i] \cdot ETC[j][i]}{(ETC)_{m(a)} \cdot \#(Res(a))}, \quad (1.37)$$

where  $Res(a)$  is the set of the tasks of the user  $a$  which must be rescheduled, and  $(ETC)_{m(a)}$  is the (expected) maximal computation time of the tasks of the user  $a$  in a considered schedule, i.e.:

$$(ETC)_{m(a)} = \max_{\substack{j \in Task(a) \\ i \in machines(a)}} ETC[j][i]. \quad (1.38)$$

In **Secure mode** the users must pay the cost of the verification of the security condition for their tasks. The cost of possible failures of machines during the tasks executions are calculated as the products of the failure probabilities and the expected times of computation of the tasks on the inaccessible machines. The secure cost function  $Q_a^{(sec)}$  in this case is defined as follows:

$$Q_a^{(sec)}[secure] = \sum_{j=k_{a-1}+1}^{k_a} \frac{P_f[j][i] \cdot ETC[j][i]}{(ETC)_{m(a)} \cdot k_a}, \quad (1.39)$$

- 1: Send the `ready_times` vectors to the individual players;
- 2: Individual players compute the  $\min Q_a$  values;
- 3: Receive the  $\min Q_a$  values from the individual players;
- 4: Send the  $\min Q_a$  values to **Global Module**;

**Figure 1.4** The optimization procedure in **Player's Module**

### 1.4.2 Genetic-based Resolution Methods for Game Models

Due to multiple constraints and different preferences of the grid users, genetic-based heuristic approaches seem to be the best candidate methodologies for solving the users' games. However, differently to the classical genetic-based schedulers (see [8]), in this case the main framework of the scheduler must be extended by the hybridization of genetic algorithm (GA) working in the **Global Module** with some other heuristic method implemented in the **Players' Module** (see Fig. 1.3).

Four hybrid GA-based schedulers have been defined for solving the symmetric grid users' games (see also [20]. The combinations of the heuristic components of these hybrids are presented in Table 1.1.

**Table 1.1** Hybrid meta-heuristics for risky and secure-assured scheduling

<b>Meta-heuristic</b>	<b>Global Module</b>	<b>Players' Module</b>
<b>RGA-GA</b>	RGA	PGA
<b>RGA-PMCT</b>	RGA	PMCT
<b>SGA-GA</b>	SGA	PGA
<b>SGA-PMCT</b>	SGA	PMCT

The GA-based meta-heuristics may work as global and local optimizers in the **Global** and **Players'** Modules. Similarly to the methodologies presented in [20], each hybrid algorithm is defined as a combination of two methods, namely *Risky Genetic Algorithm (RGA)* and *Secure Genetic Algorithm (SGA)*– in the **Global Module**; and two local level optimizers, namely *Player's Genetic Algorithm (PGA)* and *Player's Minimum Completion Time (PMCT)*– in the **Players' Module**. The performance of the **Players'** Module can be improved by receiving from the **Global** Module and updating just the changes in machine completion times for each player. Therefore the general procedure of **Players'** algorithm may be defined as it is demonstrated in Fig. 1.4.

**Schedulers Implemented in Global Module.** The template of the main GA-based engine implemented in **Global Module** is presented in Fig 1.5.

The main difference between *RGA* and *SGA* algorithms is the method of the evaluation of the population by using the users' cost functions  $Q_a$ , that is different in the **Risky** (RGA) and **Secure** (SGA) scheduling scenarios.

```

1: Generate the initial population  $P^0$  of size  $\mu$ ;
2: Send the ready times vectors of the machines corresponding to the individuals
   of the population  $P^0$  to the Player's Module;
3: Receive the  $\min Q_a$  values from the Player's Module;
4: Evaluate  $P^0$ ;
5: while not termination-condition do
6:   Select the parental pool  $T^t$  of size  $\lambda$ ;  $T^t := \text{Select}(P^t)$ ;
7:   Perform crossover procedure on pairs of individuals in  $T^t$  with probability  $pc$ ;  $P_c^t := \text{Cross}(T^t)$ ;
8:   Perform mutation procedure on individuals in  $P_c^t$  with probability  $pm$ ;  $P_m^t := \text{Mutate}(P_c^t)$ ;
9:   Send the ready times vectors of the machines corresponding to the individuals
   of the population  $P_m^t$  to the Player's Module;
10:  Receive the values  $\min Q_a$  from the Players' Module;
11:  Evaluate  $P_m^t$ ;
12:  Create a new population  $P^{t+1}$  of size  $\mu$  from individuals in  $P^t$  and/or  $P_m^t$ ;
13:   $t := t + 1$ ;
14: end while
15: return Best found individual as solution;

```

**Figure 1.5** Genetic Algorithm template

```

1: Receive the population of schedules and ready_times of the machines from the Global Module;
2: for all Schedule in the population do
3:   Calculate the completion times of the machines in a given schedule;
4:   for all Individual user  $a$  do
5:     for all User's Task do
6:       Find the machine that gives minimum completion time;
7:       Assign task to its best machine;
8:       Update the machine completion time;
9:     end for
10:    Calculate the  $\min Q_a$  value for a given schedule;
11:  end for
12:  Send the  $\min Q_a$  values to the Global Module;
13: end for

```

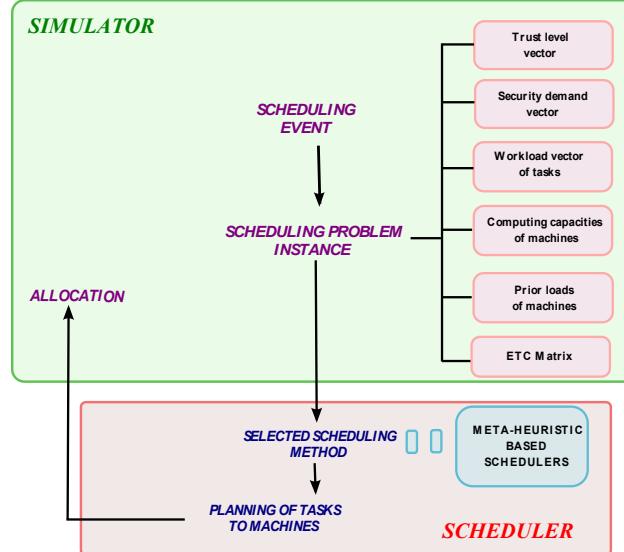
**Figure 1.6** PMCT algorithm template

**Local Schedulers in Players' Module.** Schedulers implemented in the **Players' Module** should be simple and fast in the optimization of the players' cost functions. In fact each of these functions depends only on the decision variables of the user, for whom the function was defined. For the purpose of simple case study presented in this chapter, we have implemented two heuristic grid schedulers, namely *Player's Genetic Algorithm (PGA)* and *Player's Minimum Completion Time - (PMCT)*.

*Player's Genetic Algorithm* is a simple extension of the classical GA-based scheduler defined in Fig. 1.5 applied independently for each user  $a$  with the cost function  $Q_a$  as the fitness measure. The genetic operations are executed on sub-schedules of the length  $k_a$  labeled just by the tasks submitted by user  $a$ . *PMCT* algorithm is a modification of *Minimum Completion Time - MCT* method. In this method, a task is assigned to the machine yielding the earliest (shortest) completion time. The process is repeated until all tasks are scheduled. The template of the main mechanism of *PMCT* procedure is defined in Fig. 1.6.

### 1.4.3 Empirical Setup

Four hybrid meta-heuristics defined in Table 1.1 have been integrated with the *Sim-G-Batch* simulator in order to reflect the various grid scenarios. *Sim-G-Batch* simulator is an event based grid tool that allows an easy adaptation to all dynamical changes of the systems and tasks states and characteristics. The main concept of the security-aware version of *Sim-G-Batch* simulator is presented in Fig. 1.7.



**Figure 1.7** General Flowchart of the secure *Sim-G-Batch* Simulator Linked to Scheduling

The *Sim-G-Batch* simulator generates an instance of the scheduling problem by using the following input data:

- workload vector of tasks ,
- computing capacity vector of machines,
- vector of prior loads of machines,
- *ETC* matrix of estimated execution times of tasks on machines, and
- trust level and security demand vectors for machines and tasks in a given batch.

The capacity of the resources and the workload of tasks are randomly generated by using the Gaussian distribution [23]. It is also assumed that all tasks submitted to the system must be scheduled and all machines in the system can be used.

The structure of the *Sim-G-Batch* application is based on the 2-module *HyperSim-G* simulator architecture [36] and it is composed of *Simulator* and *Scheduler* modules. The instance of the scheduling problems generated by the *Simulator* are passed on to

**Table 1.2** Values of key parameters of the grid simulator in static and dynamic cases

	<b>Small</b>	<b>Medium</b>	<b>Large</b>	<b>Very Large</b>
<b>Static case</b>				
<i>Nb. of machines</i>	32	64	128	256
<i>Resource cap.</i> (in MHz CPU)		$N(5000, 875)$		
<i>Total nb. of tasks</i>	512	1024	2048	4096
<i>Workload of tasks</i>		$N(250000000, 43750000)$		
<i>Security demandss<sub>j</sub></i>		$U[0.6; 0.9]$		
<i>Trust levels tl<sub>i</sub></i>		$U[0.3; 1]$		
<i>Failure coefficient α</i>		3		
<b>Dynamic case</b>				
<i>Init. nb of machines</i>	32	64	128	256
<i>Max nb of machines</i>	37	70	135	264
<i>Min nb of machines</i>	27	58	121	248
<i>Resource cap.</i> (in MHz CPU)		$N(5000, 875)$		
<i>Add machine</i>	$N(625000, 93750)$	$N(562500, 84375)$	$N(500000, 75000)$	$N(437500, 65625)$
<i>Delete machine</i>		$N(625000, 93750)$		
<i>Max nb of tasks</i>	512	1024	2048	4096
<i>Init. nb of tasks</i>	384	768	1536	3072
<i>Workload</i>		$N(250000000, 43750000)$		
<i>Security demandss<sub>j</sub></i>		$U[0.6; 0.9]$		
<i>Trust levels tl<sub>i</sub></i>		$U[0.3; 1]$		
<i>Failure coefficient α</i>		3		

the *Scheduler*, where the scheduling method selected and activated. The *Scheduler* generates the optimal schedules according to the specified scheduling criteria and sends the schedules back to the *Simulator*. The *Simulator* makes the allocation of the grid resources and re-schedules any tasks assigned to machines which are unavailable in the system.

Table 1.2 presents the key input parameters of the simulator in static and dynamic cases in four considered grid size scenarios, namely *Small*, *Medium*, *Large* and *Very Large* in static and dynamic modes<sup>3</sup>. Most of those parameters (excluding the numbers of tasks and machines) were tuned in empirical analysis presented in [29], [30], [21] and some recent publications of the authors of this chapter on security-aware grid scheduling supported by the game-theoretical models [5], [19], [20].

There are 16 players in the game. The parameters of all genetic schedulers working in **Global** and **Players'** Modules are defined in Table 1.3.

<sup>3</sup>The notation  $N(a, b)$  and  $E(c, d)$  is used for Gaussian and exponential probability distributions.

**Table 1.3** GA settings in the **Global** and **Players' Modules** for large static and dynamic benchmarks

Parameter	Global Module	Players' Module
number of executed genetic epochs	$(5 * (n))$	$(\lceil 0.5 * (n) \rceil)$
population size ( <i>pop_size</i> )	60	20
intermediate pop.	48	14
selection method	LinearRanking	
crossover method	Cycle (CX)	
cross probab.	0.8	0.8
mutation method	Rebalancing	
mutation probab.	0.2	
initialization	LJFR-SJFR + Random	
<i>max_time_to_spend</i>	500 secs (static) / 800 secs (dynamic)	

The combination of the genetic operators presented in Table 1.3 has been specified based on the empirical tuning process provided in [19], [18]. In *LJFR-SJFR + Random* initialization method all by two individuals are selected randomly. These two individuals are generated by using the *Longest Job to Fastest Resource - Shortest Job to Fastest Resource* (*LJFR-SJFR*) heuristic [35], where initially the number of  $m$  tasks with the highest workload are assigned to the available  $m$  machines sorted in ascending order by the computing capacity criterion. Then the remaining unassigned tasks are allocated to the fastest available machines. In *Linear Ranking* method a selection probability for each individual in a population is proportional to the rank of the individual. The rank of the worst individual is defined as zero, while the best rank is defined as  $pop\_size - 1$ , where  $pop\_size$  is the size of the population. In *Cycle Crossover (CX)* [24], first, a cycle of alleles is identified. The crossover operator leaves the cycles unchanged, while the remaining segments in the parental strings are exchanged. In *Rebalancing* mutation method, first, the most overloaded machine is selected. Two tasks  $j$  and  $\hat{j}$  are identified in the following way:  $j$  is assigned to another machine  $i'$ ,  $\hat{j}$  is assigned to  $i$  and  $ETC[j][i'] \leq ETC[\hat{j}][i]$ . Then the assignments are interchanged for tasks  $j$  and  $\hat{j}$ .

**Performance Measures.** The performances of all schedulers in empirical analysis were evaluated under the following three metrics:

- *Makespan* – the dominant scheduling criterion;
- *Flowtime* – the second (in the hierarchy) scheduling criterion; and
- *FailureRate*  $Fail_r$  parameter defined as follows:

$$Fail_r = \frac{n_{failed}}{n} \cdot 100\% \quad (1.40)$$

where  $n_{failed}$  is the number of unfinished tasks, which must be rescheduled.

Both *Makespan* and *Flowtime* measures are expressed in arbitrary time units specified for the scheduling.

#### 1.4.4 Results

Each experiment was repeated 30 times under the same configuration of parameters and operators.

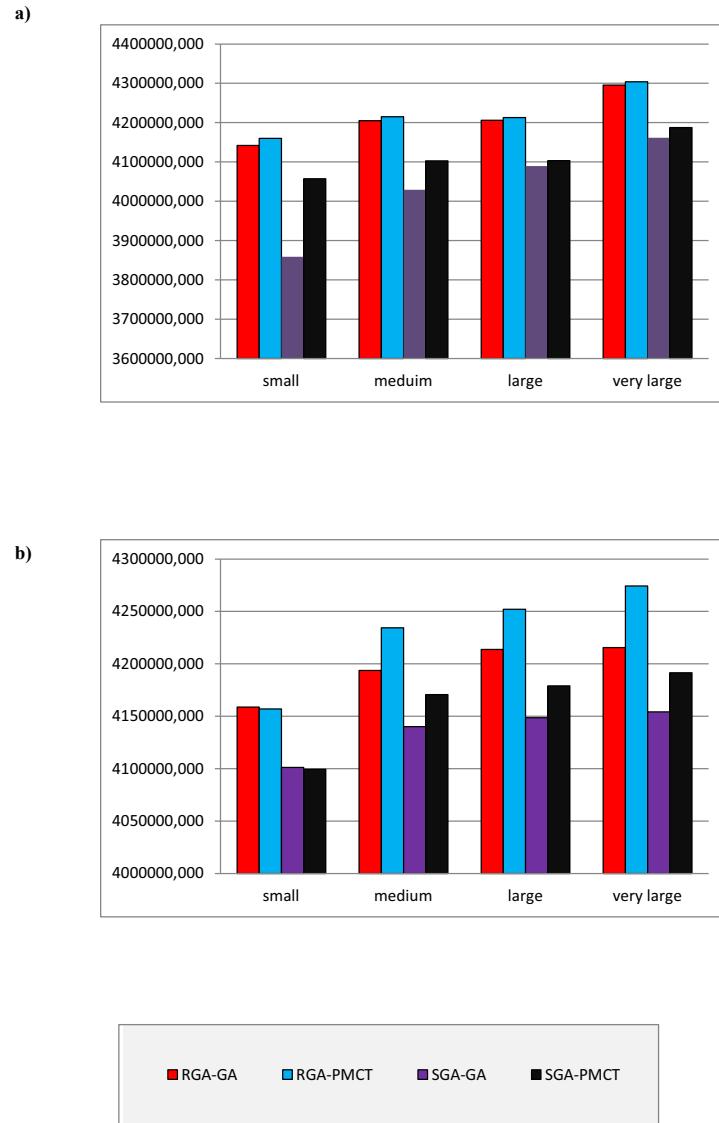
The histograms of the average values of *Makespan* and *Flowtime* achieved by four hybrid meta-heuristics designed for solving the users game are presented in Fig. 1.8 and Fig. 1.9.

Table 1.4 presents the comparison of the average values of failure rate  $Fail_r$  parameter for all considered meta-heuristics

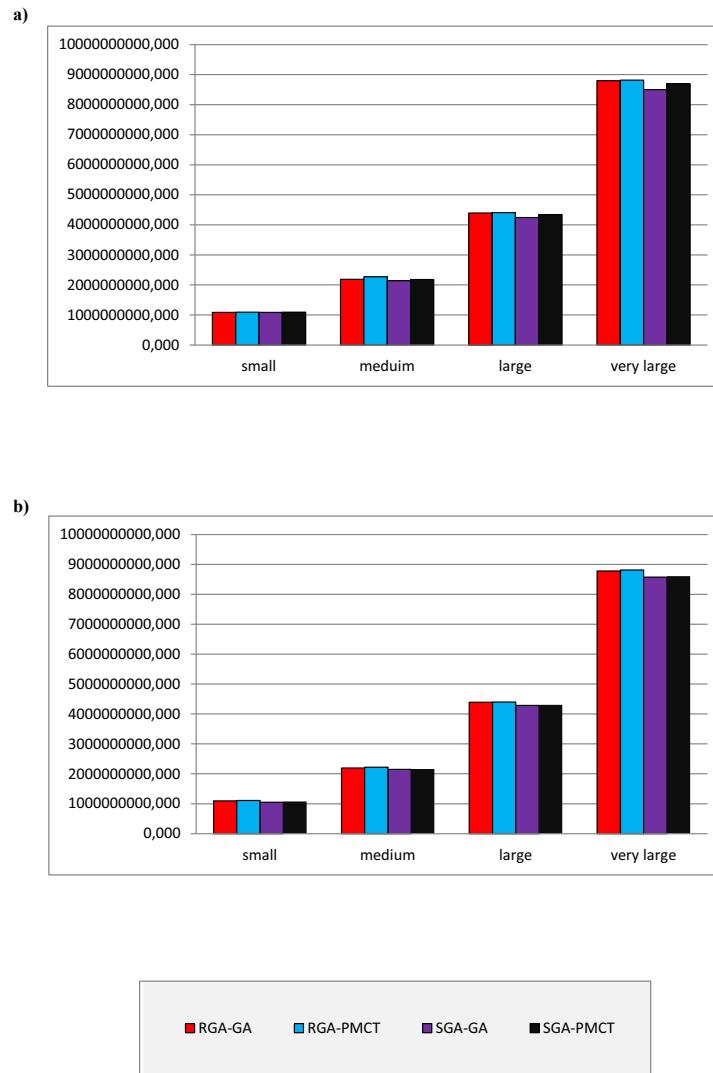
**Table 1.4** Average values of failure rate  $Fail_r$  parameter for four genetic schedulers [ $\pm s.d.$ ], ( $s.d.$  = standard deviation)

Strategy	Small	Medium	Large	Very Large
<b>Static Instances</b>				
<b>RGA-GA</b>	4.278% [ $\pm 0.99$ ]	4.766% [ $\pm 1.80$ ]	6.976% [ $\pm 1.34$ ]	7.522% [ $\pm 1.89$ ]
<b>RGA-PMCT</b>	3.993% [ $\pm 1.04$ ]	4.089% [ $\pm 1.98$ ]	<b>6.436%</b> [ $\pm 1.35$ ]	7.336% [ $\pm 2.23$ ]
<b>SGA-GA</b>	3.877% [ $\pm 0.98$ ]	4.356% [ $\pm 1.26$ ]	6.543% [ $\pm 1.89$ ]	8.015% [ $\pm 1.53$ ]
<b>SGA-PMCT</b>	<b>3.738%</b> [ $\pm 0.92$ ]	<b>4.005%</b> [ $\pm 1.05$ ]	6.456% [ $\pm 1.35$ ]	<b>6.832%</b> [ $\pm 1.56$ ]
<b>Dynamic Instances</b>				
<b>RGA-GA</b>	5.904% [ $\pm 1.24$ ]	8.469% [ $\pm 1.56$ ]	8.546% [ $\pm 1.42$ ]	9.546% [ $\pm 1.67$ ]
<b>RGA-PMCT</b>	4.880% [ $\pm 0.98$ ]	6.097% [ $\pm 1.62$ ]	7.456% [ $\pm 1.32$ ]	<b>7.126%</b> [ $\pm 2.11$ ]
<b>SGA-GA</b>	4.423% [ $\pm 0.73$ ]	5.533% [ $\pm 0.69$ ]	6.944% [ $\pm 0.98$ ]	7.354% [ $\pm 1.44$ ]
<b>SGA-PMCT</b>	<b>3.875%</b> [ $\pm 0.88$ ]	<b>4.542%</b> [ $\pm 1.03$ ]	<b>5.953%</b> [ $\pm 1.21$ ]	7.211% [ $\pm 1.95$ ]

The best results for *Makespan* and *Flowtime* in all considered grid scenarios were achieved by *SGA-GA* scheduler. Especially in static “Small” grid this method is very effective in *Makespan* reduction. The differences in the *Flowtime* results achieved by all hybrid meta-heuristics are not so significant, while in the case of *Makespan* both *SGA* hybrids significantly outperform **risky** hybrids in all grid scenarios. The lowest failure rates were observed for another hybrid of *SGA*, with the *PMCT* algorithm in the **Players’ Module**. This method achieved the best results in all but two instances.



**Figure 1.8** Empirical average *Makespan* results for non-cooperative symmetric game: (a) in static case, (b) in dynamic case.



**Figure 1.9** Empirical average *Flowtime* results for non-cooperative symmetric game: (a) in static case, (b) in dynamic case.

The results of those simple experiments lead to the following conclusion: *although the security requirements would imply some additional cost to the users of the grid system, it is worth assuming this cost in order to allocate tasks to trustful resources.*

## 1.5 LESSON LEARNT

This chapter illustrated various security aware scheduling scenarios in today's grid systems, where the autonomous decisions and individual strategies of the users have a great impact on the optimization of the cumulative costs of scheduling their tasks. Game-theoretic models have been characterized as the effective methodologies for supporting the grid users' decisions in the large-scale dynamic grid environment. The users' behavior can be effectively translated into the computational model linked to the grid scheduling.

The procedures of simulation and solving even very simple users' games may be very complex, mainly due to the high system dynamics and large amount of strategic parameters that must be specified. However, our simple empirical case study showed the high efficiency of heuristic-based resolution methods for the considered game-based models, especially in the case of additional security costs paid by the users. We believe that for variety of the real-life game scenarios, the game-based model concepts can be successfully implemented also in cloud environments, where the secure scheduling and information management remain still open and challenging research problems.

## REFERENCES

---

1. A. Abraham, R. Buyya, and B. Nath. Nature's heuristics for scheduling jobs on computational grids. In *Proc. of the 8th IEEE International Conference on Advanced Computing and Communications, India*, pages 45–52, 2000.
2. S. Ali, H.J. Siegel, M. Maheswaran, S. Ali, and D. Hensgen. Task execution time modeling for heterogeneous computing systems. In *Proc. of the Workshop on Heterogeneous Computing*, pages 185–199, 2000.
3. F. Azzedin and M. Maheswaran. Integrating trust into grid resource management systems. In *Proc. Int. Conf. Parallel Processing*, 2002.
4. T. Baçsar and G.J. Olsder. *Dynamic Non-cooperative Game Theory*. Academic Press, London, 2nd edition, 1995.
5. M. Bogdański, J. Ko“ odziej, and F. Xhafa. Supporting the security awareness of game-based grid schedulers by artificial neural networks. In *Proc. of 'International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS 2011), Seoul, 30.06-2.07.2011*, pages 277–284. IEEE Soc. Press, Los Alamitos.
6. E. Cody, R. Sharman, R. H. Rao, and S. Upadhyaya. Security in grid computing: A review and synthesis. *Decision Support Systems*, Vol. 44:749–764, 2008.
7. L.E. Edlefsen and C.B. Millham. On a formulation of discrete n-person non-cooperative games. *Metrika*, Vol. 18, No 1:31–34, 1972.
8. Y. Gao, H. Rong, and J.Z. Huang. Adaptive grid job scheduling with genetic algorithms. *Future Generation Computer Systems*, Vol. 21, No. 1:151–161, 2005.

9. S.K. Garg, R. Buyya, and H.J. Segel. Scheduling parallel applications on utility grids: Time and cost trade-off management. 2009.
10. P. Ghosh, N. Roy, K. Basu, and S.K. Das. A game theory based pricing strategy for job allocation in mobile grids. In *Proc. of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS 04), Santa Fe, New Mexico*, 2004.
11. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, Vol. 5:287–326, 1979.
12. S. Hotovy. Workload evolution on the cornell theory center IBM SP2. In *Proc. of the Workshop on Job Scheduling Strategies for Parallel, IPPS'96*, pages 27–40, 1996.
13. M. Humphrey and M.R. Thompson. Security implications of typical grid computing usage scenarios. In *Proc. of the Conf. on High Performance Distributed Computing*, 2001.
14. J. Kacprzyk. *Multistage Decision Making under Fuzziness*. Verlag TUV, Rheinland, Cologne, 1983.
15. S. U. Khan and I. Ahmad. A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids. *IEEE Tran.on Parallel and Distributed Systems*, Vol. 20, No. 3:346–360, 2009.
16. S.U. Khan and I. Ahmad. Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation. In *Proceedings of International Parallel and Distributed Proceedings Symposium (IPDPS 2006)*, pages 101–104, 2006.
17. J. Ko“odziej and F. Xhafa. A game-theoretic and hybrid genetic meta-heuristic model for security-assured scheduling of independent jobs in computational grids. In *Proc. of CISIS 2010, Cracow, 15-18.02.2010, in L. Barolli, F. Xhafa and S. Venticinque eds.*, pages 93–100, 2010.
18. J. Ko“odziej and F. Xhafa. Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population. *Future Generation Computer Systems*, 27:1035–1046, 2011.
19. J. Ko“odziej and F. Xhafa. Meeting security and user behaviour requirements in grid scheduling. *Simulation Modelling Practice and Theory*, Vol. 19, No. 1:213–226, 2011.
20. J. Ko“odziej and F. Xhafa. Modern approaches to modelling user requirements on resource and task allocation in hierarchical computational grids. *Int. J. on Applied Mathematics and Computer Science*, Vol. 21, No. 2:243–257, 2011.
21. Y.-K. Kwok, K. Hwang, and S. Song. Selfish grids: Game-theoretic modeling and nas/psa benchmark evaluation. *IEEE Tran. on Parallel and Distributing Systems*, Vol. 18, No. 5:1–16, 2007.
22. G. Laccetti and G. Schmidb. A framework model for grid security. *Future Generation Computer Systems*, Vol. 23 (2007):702–713, 2007.
23. P. S. Mann. *Introductory Statistics*, 7th ed. Wiley, 2010.
24. I Olivier, D. Smith, and J. Holland. A study of permutation crossover operators on the travelling salesman problem. In *Proc. of the ICGA 1987, Cambridge, MA*, pages 224–230, 1987.
25. N.G. Pavlidis, K.E. Parsopoulos, and M.N. Vrahatis. Computing nash equilibria through computational intelligence methods. *Journal of Computational and Applied Mathematics*, Vol. 175:113–136, 2005.

26. A. Pedrycz. Finite cut-based approximation of fuzzy sets and its evolutionary optimization. *Fuzzy Sets and Systems*, Vol. 160, No. 24, 2009.
27. W. Pedrycz. Statistically grounded logic operators in fuzzy sets. *European Journal of Operational Research*, Vol. 193, No. 2:520–529, 2009.
28. T. Roughgarden. Stackelberg scheduling strategies. *SIAM Journal on Computing*, Vol. 33, No. 2:332–350, 2004.
29. S. Song, K. Hwang, and Y.K. Kwok. Trusted grid computing with security binding and trust integration. *J. of Grid Computing*, Vol. 3, No. 1-2:53–73, 2005.
30. S. Song, K. Hwang, and Y.K. Kwok. Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling. *IEEE Tran. on Computers*, Vol. 55, No. 6:703–719, 2006.
31. P.D. Straffin. *Game Theory and Strategy*. Mathematical Association of America Textbooks, 1996.
32. E-G. Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, USA, 2009.
33. C.-C. Wu and R.-Y. Sun. An integrated security-aware job scheduling strategy for large-scale computational grids. *Future Generation Computer Systems*, Vol. 26 (2010):198–206, 2010.
34. F. Xhafa and A. Abraham. Computational models and heuristic methods for grid scheduling problems. *Future Generation Computer Systems*, Vol. 26 (2010):608–621, 2010.
35. F. Xhafa, J. Carretero, and A. Abraham. Genetic algorithm based schedulers for grid computing systems. *Int. J. of Innovative Computing, Information and Control*, Vol. 3, No. 5:1053–1071, 2007.
36. F. Xhafa, J. Carretero, L. Barolli, and A. Durresi. Requirements for an event-based simulation package for grid systems. *Journal of Interconnection Networks*, Vol. 8, No. 2:163–178, 2007.