2010 International Conference on Complex, Intelligent and Software Intensive Systems

A Game-Theoretic and Hybrid Genetic Meta-Heuristics Model for Security-Assured Scheduling of Independent Jobs in Computational Grids

Joanna Kołodziej
Department of Mathematics and Computer Science
University of Bielsko-Biała
ul. Willowa 2, Bielsko-Biała, Poland
Email: jkolodziej@ath.bielsko.pl

Fatos Xhafa
Department of Computer Science and Information Systems
Birkbeck, University of London
Malet Street, Bloomsbury, London WC1E 7HX, UK.
Email: fatos@dcs.bbk.ac.uk

***Abstract*—Scheduling independent tasks in Computational Grids commonly arises in many Grid-enabled large scale applications. Much of current research in this domain is focused on the improvement of the efficiency of the Grid schedulers, both at global and local levels, which is the basis for Grid systems to leverage large computing capacities. However, unlike traditional scheduling, in Grid systems security requirements are very important to scheduling tasks/applications to Grid resources. The objective is thus to achieve efficient and secure allocation of tasks to machines. In this paper we propose a new model for secure scheduling at the Grid sites by combining game-theoretic and genetic-based meta-heuristic approaches. The game-theoretic model takes into account the realistic feature that Grid users usually perform independently of each other. The scheduling problem is then formalized as a noncooperative non-zero sum game with Nash equilibria as the solutions. The game cost function is minimized, at global and user levels, by using four genetic-based hybrid meta-heuristics. We have evaluated the proposed model through a static benchmark of instances, for which we have measured two basic metrics, namely the makespan and flowtime. The obtained results suggest that it is more resilient for the Grid users (and local schedulers) to tolerate some job delays defined as additional scheduling cost due to security requirements instead of taking a risk of allocating at unreliable resources.**

***Keywords*-Computational Grids, Scheduling, Non-cooperative Games, Nash Equilibrium, Genetic Algorithms, Min-Min heuristic.**

I. Introduction

Grid Computing systems has emerged as the next generation of parallel and distributed computing systems based on large-scale infrastructures. Such systems are currently comprising different forms by targeting different objectives, namely, Computational Grids, Desktop Grids, Enterprise Grids, Scavenging Grids, Data Grids, etc. Computational Grids were the first to address the resolution of complex problems from eScience, a family of problems requiring substantially more computational power, arising in meteorology, industry, physics, medicine, finance, etc., for which Grid-enabled solutions have made possible to achieve breakthroughs in their resolution times.

Computational Grids primarily address the development of high-performance applications, running in parallel on multiple computers, clusters or super-computers connected by wide-area networks. The objective is thus to achieve high throughput, minimized makespan and flowtime, among others. Such applications are usually parallel in nature, that is, they can be split in many independent or loosely coupled tasks. For instance, in parameter sweep applications [Casanova et al., 2000], which arise in many scientific and engineering fields such as Computational Fluid Dynamics and Particle Physics, many tasks perform similar computations for varying input parameters over large parameter spaces. Despite the easy parallelization of such applications, achieving the high performance goal is challenging in Computational Grids. Indeed, Computational Grids virtually join large amount of computational resources but the high degree of heterogeneity of resources and that of interconnection networks makes it difficult to achieve high performance applications. Scheduling is here a must in order to cope in practice with the heterogeneity and dynamic nature of such systems.

Most current efforts to scheduling on distributed systems such as Computational Grids are devoted to achieving the maximum throughput from the entire system. Development in Grid systems are having each time more requirements beyond the high performance Grid. One such requirement, which only recently has just started to be investigated, is that of security. Consider for example the following scenario. A large bank accounting for millions of costumers wants to periodically process the huge log file of its costumers activity in the online banking system for detecting aborted transactions or for studying costumers' behavior within the online banking system. The application for processing the large log file can certainly be easily parallelized and processed in a Grid system since log files usually follow regularly sequence data structuring. However, the Bank is not only concerned with the efficiency of processing for meeting delivery deadline, but also with a security level while processing the data on computational nodes of the Grid system, which could belong to different administrative domains. Therefore, the Grid scheduler not only should compute an efficient planning of tasks to Grid nodes but

978-0-7695-3967-6/10 $26.00 © 2010 IEEE
DOI 10.1109/CISIS.2010.74

also should allocate the tasks to those nodes that assure the requested level of security. Integrating desired security levels into scheduling is thus very important for Grid-enabled applications. Standard security mechanisms such as sandboxing as well as conventional authentication methods offered by grid middleware might not be adequate to prevent from security threats.

Secure scheduling has been recently addressed in the Grid computing literature. In [Song et al., 2006], the authors consider the risk and insecure conditions in Grid task scheduling caused by software vulnerability and distrusted security policy. Risk-resilient scheduling algorithms were proposed to assure secure Grid task execution under such risky conditions. Azzedin and Maheswaran [Azzedin and Maheswaran, 2002], presented a trust model for Grid systems and used it to incorporate security requirements into scheduling algorithms; trust-aware heuristic scheduling were developed by using known heuristics, namely, Min-Min, Minimum Completion Time and Sufferage methods.

In this paper we propose a new model for secure Grid scheduling by combining game-theoretic and meta-heuristic approaches. We use game-theoretic model to formalize the scheduling problem as a non-cooperative game. The rationale behind is that in most realistic scenarios, Grid users are independent, that is, they independently submit their tasks/application to the Grid system. The game cost function is minimized at global and user levels by using four hybrid heuristics combining GAs and modified Min-Min method. The proposed model has been evaluated through a static benchmark of instances, for which we have measured two metrics, namely makespan and flowtime.

The rest of the paper is organized as follows. In Section II we introduce a few preliminary concepts on independent task scheduling and game-theoretic models, specifically on non-cooperative games. We present the game-theoretical model of security-assured scheduling in Section III. The experimental evaluation using a static benchmark is given in Section IV. We end the paper in Section V with some conclusions and indications for future work.

II. Preliminaries

A. Independent scheduling in Computational Grids

Independent scheduling in distributed systems is the version of scheduling that computes a planning of a given set of independent tasks to a set of available machines. The requirement over tasks to be independent (in some cases, tasks can be considered loosely coupled) is the main characteristics. This kind of scheduling is very suitable to address in large scale Grid systems for many reasons. The absence of dependencies among tasks makes it easier to preemptive or re-schedule tasks. It will not affect directly completion of other tasks, or, in case it is not possible to migrate a task to available machines, the task can be re-scheduled again. Also the resource characteristics can be better exploited as independent tasks could vary on the computation grain (some could be coarse grain and some others fine grain).

Beyond the need of ensuring high performance Grid-enabled applications, usually through the makespan, Grid scheduling seeks also to maximize Grid resource utilization as well as to achieve QoS of the Grid system, e.g. through the flowtime of the system. Resource utilization is particularly interesting for Grid systems since such systems could be conceived as contributory systems and thus incentiveness to owners' resources is important. QoS is important in terms of response time to users who submit tasks/applications to the Grid system.

In this work we consider the problem formulation based on the Expected Time to Compute matrix model. In this model [Ali et al., 2000], it is assumed that we know:

- The estimation of the computational load of each task (e.g. in millions of instructions). This information is usually given from task specification or is extracted from execution traces.
- The computing capacity of each machine (e.g. in millions of instructions per second, MIPS).
- The estimation of the prior load of each one of the available machines.
- The entries of the *ETC* matrix, denoted by $ETC[t][r]$ indicates the expected time to compute task t in resource r. A simple way to compute $ETC[t][r]$ entries is by dividing the computing load of task t by the computing capacity of machine r.

B. Game-theoretical models and non-cooperative games

Game theory is playing an important role in computer science, where it is being used as a means for modeling interactive computations or multi-agent systems. Recently, Internet computing is seen as a new domain of applications of game theory, which in combination with economic theory can develop algorithms for finding equilibria in computational markets, computational auctions, Grid and P2P systems as well as security and information markets.

An important challenge in using game-theoretic models for Grid scheduling is the large size scale of the Grid system and the fact that resources cross different administrative domains. Specifically, the hierarchical relationship among computers in Internet, namely, global level, inter-site level and intra-site level should be translated to the game-theoretic model [Kwok et al., 2007].

One important class of game-theoretic models is that of non-cooperative games, in which players make decisions independently. Due the nature of large scale Grid systems, in which cooperation is difficult to happen at large scale, this kind of game-theoretic model is a potential model for integrating security in Grid scheduling.

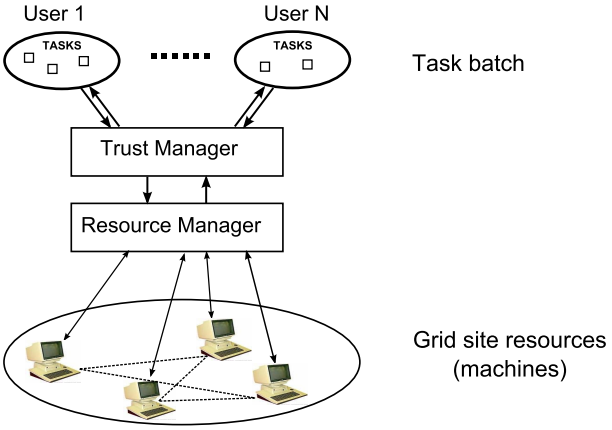

Figure 1. The model of security-assured Grid site.

III. Game-theoretical model of security-assured scheduling at Grid site

The security-assured site in Computational Grid can be modeled as a network of m heterogeneous machines, for which $TrustLevels$ (TL) are defined. The TL parameters are analyzed by a trust manager and they specify how much user can trust a local resource manager which maintains machines status and monitors tasks execution. A task can be successfully completed when a *security assurance condition* is satisfied, i.e. $(TL \geq SD)$, where SD is a $SecurityDemand$ of the task issued to all available machines. Both TL and SD parameters were introduced by Song et al. in [Song et al., 2006]. Similarly as in their work $TrustLevel$ can be understood as an aggregation of the site behavior and intrinsic security attributes such as intrusion detection or firewall. The process of matching TL with SD reminds the real-life case where the users of some portals like Yahoo! are required to specify the security level of the login session. The task failure at the Grid site is usually the result of its inaccessibility from a security barricade.

A concept of a simple model of security-assured Grid site is presented in Fig. 1

In this work we consider the Independent Job Scheduling problem, in which tasks are processed in the batch mode [Xhafa et al., 2007 b)]. The total number of tasks in the batch, denoted by n, can be calculated as the sum of tasks submitted by all users, i.e.:

$$n = \sum_{l=1}^{N} k_l, \tag{1}$$

where N is the number of Grid users and k_l is the number of tasks submitted by the user l for execution at the site resources.

A *schedule* of the batch of tasks at the Grid site is defined as a vector x:

$$x = [x_1, \ldots, x_{k_{l-1}}, \ldots, x_{k_l}, \ldots, x_n]^T, \tag{2}$$

where $x_j \in [1, m]$ indicates the number of the machine, to which task j is assigned $(j = 1, \ldots, k_{l-1}, \ldots, k_l, \ldots, n)$, n is the total number of tasks and m is the number of available machines.

A. Nash strategy for non-cooperative users

Let us assume that Grid users cannot cooperate with each other. Each of them tries to choose an optimal strategy of the allocation of his tasks to machines in order to minimize the total cost of the scheduling of the batch of tasks. We can then define the scheduling problem at the security-assured Grid site as an N-player non-cooperative game denoted by $G_N = (N; \{J_l\}_{l=1,\ldots N}; \{Q_l\}_{l=1,\ldots,N})$, where:

- N is the number of players (Grid users),
- $\{J_1, \ldots, J_N\}; l = 1, \ldots, N$ are the sets of strategies of the players,
- $\{Q_1, \ldots, Q_N\}; Q_l : J_1 \times \ldots \times J_N \rightarrow \mathbb{R}; \forall_{l=1,\ldots,N}$ is the set of users' cost functions.

The set of strategies of the user l is defined as the Cartesian product $J_l = \widehat{J_{k_{l-1}+1}} \times, \ldots, \widehat{J_{k_l}}$, where:

- $\widehat{J}_s = \{(ETC[s][x_s], P_f[s][x_s]) : k_{l-1} + 1 \leq s \leq k_l$; $1 \leq x_s \leq m\}$;
- $ETC[s][x_s]$ is an element of the *ETC* matrix and indicates the expected time to compute the task s in the resource x_s;
- $P_f[s][x_s]$ is an element of a *Task Failure Probability* matrix (TFP).

We denote by $P_f[s][x_s]$ the probability of the failure of the task s on machine x_s which is modeled by an exponential distribution given by the following formula:

$$P_f[j][x_j] = \begin{cases} 0 & , \quad s_j \leq t_{x_j} \\ 1 - e^{-\alpha(s_j - t_{x_j})} & , \quad s_j > t_{x_j} \end{cases} \tag{3}$$

The failure coefficient α is a fraction number. The negative exponent indicates failure grows with the difference $(s_j - t_{x_j})$, where $s_j, (j = 1, \ldots, n)$, are the coordinates of the security demand vector for all tasks submitted to the Grid site denoted by $\widehat{SD} = [s_1, \ldots, s_n]$ and t_{x_j} are the coordinates of a trust level vector for the machines denoted by $\widehat{TL} = [t_1, \ldots, t_m]$. The values of s_j and t_{x_j} are real fractions in the range [0,1] with 0 representing the lowest and 1 the highest security requirements and the most risky and fully trusted machine, respectively. The trust manager is responsible for the verification of the security assurance condition for a given task-machine pair. A task j can fail on machine i if this condition is not satisfied, i.e. $s_j \geq t_i$.

The elements of the set J_l can be then defined as meta-vectors x^l, such that

$$x^l = \begin{bmatrix} ETC[k_{l-1}+1][x_{k_{l-1}+1}]; P_f[k_{l-1}+1][x_{k_{l-1}+1}] \\ \vdots \\ ETC[k_l][x_{k_l}]; P_f[k_l][x_{k_l}] \end{bmatrix} \tag{4}$$

The coordinates of the vector x^l are the decision variables of the user l. The game defined above is a special instance of a discrete N-persons game, the solution of which can be Nash equilibrium (see [Edlefsen and Millham, 1972]).

The users' costs of jobs scheduling at the security-assured Grid site cannot be limited to the costs of execution of their tasks and the utilization of the machines. The users have to "pay" an additional "fee" for the secure allocation of their tasks in the machines. Thus the functions Q_l for all $l \in \{1, \ldots, N\}$ can be defined as the sum of the following three components:

$$Q_l = Q_l^{(e)} + Q_l^{(u)} + Q_l^{(s)}, \tag{5}$$

where:

- $Q_l^{(e)}$ indicates user's task execution cost,
- $Q_l^{(u)}$ denotes a resource utilization cost, and
- $Q_l^{(s)}$ is the cost of security-assured allocation of the user's task.

The methods of the calculation of the values of the functions $Q_l^{(e)}$, $Q_l^{(u)}$ and $Q_l^{(s)}$ are specified below:

Job execution cost: A job submitted by the Grid user l is defined as a set of k_l independent tasks. The total cost of the execution of the user's job can be calculated as the sum of the expected times of the computation of the user's tasks on the machines, to which they are assigned. Thus the function $Q_l^{(e)}$ is defined using the following formulae:

$$Q_l^{(e)} = \sum_{j=k_{l-1}+1}^{k_l} ETC[j][x_j], \tag{6}$$

where $ETC[j][x_j]$ is an element of the *ETC* matrix.

Resource utilization cost: The Grid user utility function is often reduced to the cost of the resource utilization. It could be defined as a cost of buying free CPU cycles [Garg et al., 2009]. In our model the utilization cost for the user l is defined using the following formula:

$$Q_l^{(u)} = \sum_{j=k_{l-1}+1}^{k_l} (loc\_makespan - completion[x_j]) \tag{7}$$

where $completion[x_j]$ denotes the completion time of the machine x_j in a given schedule x and is calculated in the following way:

$$completion[x_j] = ready[x_j] + \sum_{\substack{p \in \{1,\ldots,n\}: \\ x[p]=x_j}} ETC[p][x_j] \tag{8}$$

A local makespan, denoted by $loc\_makespan$ in (7), can be expressed as a maximal completion time in the schedule x; locality refers to the optimality of the schedule with respect to tasks assignment of user l. Note also that $ready[x_j]$ in (8) is used for the notation of the time of finishing the execution of tasks previously assigned to the machine x_j, that is when the machine is expected to have completed prior load.

It follows from Eq. (7) that the utilization cost is minimal in the case of allocation of the user's task in the machine with the maximal completion time, which satisfies the main resource utilization criterion.

Security-assurance cost: The value of the security-assurance cost paid by the Grid user depends on the scheduling strategy and the result of the verification of security condition by the trust manager in a Grid site. We propose two scheduling strategies, known as *risky* and *preemptive* mode, which were applied also in [Song et al., 2006].

In the risky mode, all risky and failing conditions of resources are ignored by the users. The total cost of scheduling of the user's tasks can be increased by the addition of the values of $Q_l^{(s)}$ function defined as follows:

$$Q_l^{(s)} = \sum_{j=k_{l-1}+1}^{k_l} P_f[j][x_j] \cdot ETC[j][x_j]. \tag{9}$$

In the preemptive mode, a task can fail on the machine with the probability defined by (3) meeting too restricted security requirements. If a failure is observed the task will be migrated to the next available machine. If another failure is observed, the task will be migrated to the next machine until the task is successfully executed or all machines have been exhausted. In the worst case, the scheduling of the task is aborted and it can be resubmitted as a new one in the next batch of jobs. A Grid user has to "pay" for each task failure. A failure cost can be calculated as a product of the failure probability and the expected time of computation of the task on a inaccessible machine. The security-assurance cost function for the user l can be then defined in the following way:

$$Q_l^{(s)} = \sum_{j=1}^{k_l} \sum_{p=1}^{m_{(k,j)}} (P_f[k_{l-1}+j][x_p] \cdot ETC[k_{l-1}+j][x_p]), \tag{10}$$

where $m_{(k,j)}$ is the number of task failures ($j = 1, \ldots, k_l$).

Each Grid user tries to minimize his cost function in the game. Let us denote by $minQ_l, (l = 1, \ldots N)$, the minimal value of the function Q_l calculated in the following way:

$$minQ_l = \min_{x^l \in J_l} \{Q_l(x_1, ..., x_n)\} \tag{11}$$

The objective of the players of the game is to minimize *a game cost function* $Q : J_1 \times \cdots \times J_N \to \mathbb{R}$ defined in the following formulae:

$$Q(x_1, ..., x_n) = \sum_{l=1}^{N} ([Q_l(x_1, ..., x_n) - minQ_l]) \tag{12}$$

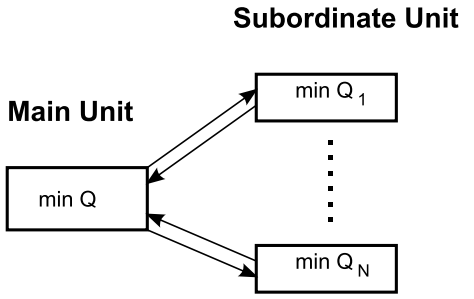

Figure 2. Two-steps procedure of finding of the Nash equilibria in non-cooperative N-person game.

The game cost function has non-negative values and the result of its global minimization is the Nash equilibrium ([Pavlidis et al., 2005], [Edlefsen and Millham, 1972]). The Nash equilibrium can be interpreted as a steady state of a strategic game, in which each player holds correct expectations concerning the other players behavior.

B. Optimization approach for solving the Grid users' game

The problem of finding the Nash equilibria of a finite strategic game remains challenging especially in a real-life approaches. It follows from Eq. (12) that we need to minimize first the cost functions of all players and then we can compute the values of the function Q. Thus the procedure of the minimization of the game cost function in the non-cooperative N-person game is composed of two cooperated units:

- **Main unit** - which solves the problem of the global level minimization of the Q function,
- **Subordinate unit** - which solves the problems of the local level minimization of the users' cost functions Q_l.

The general schema of the Nash equilibria search strategy is presented in Fig. 2. We applied it for solving the N-matrix game with Grid users as players.

Genetic Algorithms (GAs) have proved to be one of the fastest meta-heuristics in finding the optimal schedules in the case of high resource diversification in the Grid environment and large number of tasks in the batches. We propose four GA-based hybrid meta-heuristics, introduced in Table I as the global and local optimizers in the main and subordinate units.

Table I
FOUR RISK-RESILIENT HYBRID META-HEURISTICS FOR SECURE-ASSURED JOB SCHEDULING AT THE GRID SITE.

Hybrid Meta-heuristic	Main unit	Subordinate unit
RGA-GA	RGA	QGA
RGA-Min	RGA	Min
PGA-GA	PGA	QGA
PGA-Min	PGA	Min

Each of them is defined as a combination of two GA schedulers - *Risky Genetic Algorithm (RGA)* and *Preemptive Genetic Algorithm (PGA)*- in the main unit- and two user's level local optimizers - *Queued Genetic Algorithm (QGA)* and a modified *Min-Min* heuristic(*Min*) - in the subordinate unit. The detailed analysis of all applied algorithms is presented in the following two subsections.

C. GA-based schedulers in the main unit

Several implementations of GSs for independent scheduling have been proposed in the literature. In this work, we have used the GA implementation in [Xhafa et al., 2007 a)]. The general template of that GA-based scheduler is defined in Alg. 1.

Algorithm 1 Genetic Algorithm template

1: Generate the initial population P^0 of size μ;
2: Evaluate P^0;
3: **while** not termination-condition **do**
4: Select the parental pool T^t of size λ; $T^t := Select(P^t)$;
5: Perform crossover procedure on pars of individuals in T^t with probability p_c; $P_c^t := Cross(T^t)$;
6: Perform mutation procedure on individuals in P_c^t with probability p_m; $P_m^t := Mutate(P_c^t)$;
7: Evaluate P_m^t ;
8: Create a new population P^{t+1} of size μ from individuals in P^t and/or P_m^t ; $P^{t+1} := Replace(P^t; P_m^t)$
9: $t := t + 1$;
10: **end while**
11: **return** Best found individual as solution;

Based on the results of the tuning process of GA operators performed in [Xhafa et al., 2007 a)] we used linear ranking selection, cycle crossover (CX) and re-balancing mutation as an appropriate combination of such operators for our algorithm. We also applied *LJFR-SJFR (Longest Job to Fastest Resource – Shortest Job to Fastest Resource)* as an initialization procedure and elitist generational replacement method.

The above GA implementation can be directly applied to scheduling in the risky mode, where a fitness function is defined as the game cost function Q and the security-assurance costs of the users are calculated using Eq. (9). We will denote this algorithm by *RGA* according to the notation introduced in the previous section.

The main difference between *RGA* and *Preemptive GA* is the method of the evaluation of the population. In the preemptive mode this procedure is extended by scanning the population in order to verify the security condition. If task failure is observed, the cost function of the task "owner" is updated according to Eq. (10). In case of the failure of some task on all machines, the task is removed from the

batch (and from all schedules in the population) and several algorithm parameters, like the length of the chromosomes, the number of the decision variables of the task "owner", as well as the structure of the *ETC* and *TFP* matrices are updated.

D. Local schedulers in the subordinate unit

We used two modifications of the well-known Grid schedulers for the local minimization of the players' cost functions. The first, named as *Queued Genetic Algorithm*, is a simple extension of the *RGA* applied independently for each user's cost function Q_l as fitness. The GA operations are executed on sub-schedules of the length k_l labeled just by the tasks submitted by a given user. The *RGA* procedure is executed sequentially for the queue of users.

The second method, denoted by *Min*, is a modification of *Min-Min* heuristic [Freund et al., 1998]. This method starts by computing a matrix of the users' costs of the allocation and execution of an individual task on a given machine, named *task cost* and denoted by $task\_cost[j][i]$ for any task j and machine i based on the $ETC[j][i]$, $P_f[j][i]$ and $ready[i]$ values:

$$\begin{aligned} task\_cost[j][i] &= ETC[j][i] \cdot (1 + P_f[j][i]) + \\ &+ (m_j - (ETC[j][i] + ready[i])) \end{aligned} \tag{13}$$

where $m_j = \max\{(ETC[j][i] + ready[i]); j = 1, \ldots, m\}$

For any task j assigned by a given user, the machine x_j with a minimal task cost is computed by traversing the j-th row of the task cost matrix. Then, a task k with the minimal task cost is chosen and mapped to the corresponding machine x_k (previously computed). Next, the task k is removed from set of users tasks and the values $task\_cost[j][i]$ for each j in *Tasks* and machine x_k are updated. The process is repeated while there remain tasks to be assigned by the user (see Alg. 2). An experimental evaluation on the performance of the original *Min-min* method can be found in [Xhafa et al., 2007 b)]. As can be seen from Alg. 2, a double minimum (hence the name of Min-Min method) is computed in this method by first scanning the task cost matrix through rows and then by columns.

Algorithm 2 *Min* algorithm.

```
1:  for all task j ∈ Tasks do
2:    for all machine i ∈ Machines do
3:      Compute task_cost[j][i] according to Eq. (13)
4:    end for
5:  end for
6:  repeat
7:    for all user l ∈ Users do
8:      repeat
9:        for all task j_l ∈ Task_l do
10:         compute a minimal cost task and identify the
            machine on which it is achieved;
11:       end for
12:       Select task k with the lowest value of the mini-
          mal task cost;
13:       Map k to the machine x_k for which the minimal
          task cost has been previously computed;
14:       Delete k from Task_l;
15:       Update task_cost[j][i], ∀j ∈ Tasks and i = x_k;
16:     until (Task_l ≠ φ)
17:   end for
18: until (Users ≠ φ)
```

IV. Experimental analysis for static scheduling

We have conducted a preliminary experimental evaluation of the proposed hybrid meta-heuristics implementations for the problem of minimization of the game cost function defined by Eq. (12). We found it useful to initially use a static benchmark in our experimental study. The main reason is that the dynamic schedulers can run the static methods for scheduling tasks sampled in particular batches. We defined two basic metrics for measuring the effectiveness of scheduling, namely makespan and flowtime. The objective is thus to identify which of the proposed four meta-heuristic approaches performs best in minimizing the game cost function.

A. Benchmark instances and Experimental setup

Our experiments were performed on a subset of the benchmark of static instances, which are classified into different types of ETC matrices, according to task heterogeneity, machine heterogeneity and consistency. Each instance consists of 512 tasks and 16 machines and is labelled by *u_x_yyzz.0* as in [Braun et al., 2001] (in the notation, hi means high, and lo means low):

- u means uniform distribution (used in generating the matrix).
- x means the type of consistency (c–consistent, i–inconsistent and s – semi-consistent).
- yy indicates the heterogeneity of the tasks.
- zz indicates the heterogeneity of the resources.

All 512 tasks are divided into 16 sets of 32 elements. Each set belongs to one of 16 Grid users. The settings of all general parameters are presented in Table II. Table III reports key parameters for three GAs used in experiments: RGA, PGA and QGA.

Performance Metrics Evaluated: To evaluate the scheduling performance, the following performance metrics are used:

- *Flowtime:* Let F_j denotes the time when task j finalizes. The flowtime can be calculated using the following formulae:

$$Flowtime = \sum_{j \in Task} F_j \tag{14}$$

Table II
GENERAL SIMULATION PARAMETER SETTING.

Parameter	Value setting
Total number of tasks	512
Number of machines	16
Number of users	16
Number of tasks assigned by individual user	32
Task security demand (s_j)	0.6 - 0.9, uniform distribution
Resource trust levels (t_j)	0.3 - 1.0, uniform distribution
Failure coefficient	$\alpha = 3$

Table III
SETTINGS OF GAS KEY PARAMETERS IN THE MAIN AND SUBORDINATE UNITS.

Parameter	PGA and RGA settings	QGA settings
Population size	60	40
Crossover Prob.	0.8	0.8
Mutation Prob.	0.2	0.2
Stopping criterion	3000 iterations	1000 iterations

Flowtime is usually considered as a QoS criterion as it expresses the response time to the submitted task execution requests of Grid users.

- *Makespan:* Makespan is one of the basic metrics of a Grid systems performance: the smaller the value of makespan, the faster is the execution of tasks in the Grid system. Makespan can be calculated by the following formulae:

$$Makespan = \max_{j \in Tasks} F_j \tag{15}$$

The aim is to minimize both $flowtime$ and $makespan$ over the set of all possible schedules for a Grid configuration.

B. Computational results

We present in Table IV computational results obtained for the static benchmark using the four meta-heuristic implementations. In order to avoid biased results, each experiment was repeated 30 times and averaged values of makespan and flowtime are reported.

Analysis of the results: From the average makespan values shown in Table IV we can observe that *PGA-GA* outperforms *RGA* hybrids (i.e *RGA-GA* and *RGA-Min*) for all instances and *PGA-Min* performs better only for two instances of the benchmark.

Similarly as in the case of makespan, for flowtime *PGA-GA* outperforms the two meta-heuristics designed for the scheduling in the risky mode for all types of ETC matrices and it is worse than *PGA-Min* only for two instances of the benchmark.

We can conclude from our simple experimental analysis that GA-based hybrid meta-heuristics work better in the preemptive mode than in risky mode for all three groups of instances (consistent, semi-consistent and inconsistent ETC matrices). The differences in results obtained by RGA and PGA hybrids are rather significant, while the results for the algorithms of the same class are of the same rank. It means that satisfying the security policy can be one of the crucial criterion in the optimization of the users' cost functions in our approach, and in consequence, in the approximation of an optimal schedule.

The obtained $s.d.$ values are small and show that the variation in results do not surpass the range of 1-1.5 %, which means that all algorithms are stable and the total cost of scheduling did not change significantly in each trial.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an approach for secure independent task scheduling in Computational Grids. Our approach combines game-theoretic and Genetic Algoritms-based meta-heuristics. The former is used to model the scheduling problem as a non-cooperative non-zero sum game with Nash equilibria as the solutions while the later are used to minimize game cost function, at global and user levels. We have evaluated the proposed model through a static benchmark of instances in the literature, for which we have measured the makespan and flowtime values. The computational results showed that it is more resilient for the Grid users (and local schedulers) to tolerate some job delays defined as additional scheduling cost due to security requirements instead of taking a risk of allocating at unreliable resources.

The results obtained for four hybrid meta-heuristics in our approach are worse than those achieved by some GA-based schedulers in the case of just job execution cost as objectives (see [Xhafa et al., 2007 a)]). Also the execution of the proposed algorithms takes more time comparing with standard GA-based schedulers. The main reason is that the optimization two-level procedure is complex. We want to simplify it in our future work by changing the rules of the game (allowing to make some coalitions of users) and modifying the cost functions to make them more adapted to realistic scenarios. We also plan to evaluate the proposed approach in a dynamic environment using a Grid simulator.

ACKNOWLEDGMENT

Joanna Kołodziej's research work is partially supported by $10/I/GW/2008$ ATH Grant. Fatos Xhafa's research work completed at Birkbeck, University of London, on leave from Technical University of Catalonia (Barcelona, Spain). His research is supported by General Secretariat of Universities, Ministry of Education, Spain.

REFERENCES

[Ali et al., 2000] S. Ali, H.J. Siegel, M. Maheswaran and D. Hensgen: "Task execution time modeling for heterogeneous computing systems", *Proceedings of Heterogeneous Computing Workshop* , 185–199, 2000. (HCW 2000)

Table IV
COMPARISON THE AVERAGE VALUES OF MAKESPAN AND FLOWTIME ACHIEVED BY FOUR HYBRID META-HEURISTICS (IN ARBITRARY TIME UNITS; $s.d.$ - STANDARD DEVIATION)

	Average Makespan ($\pm s.d.$)				**Average Flowtime** ($\pm s.d.$)			
Instance	**RGA-GA**	**RGA-Min**	**PGA-GA**	**PGA-Min**	**RGA-GA**	**RGA-Min**	**PGA-GA**	**PGA-Min**
u_c_hihi	7802487.68 (± 51264.12)	7856833.55 (±51016.33)	**7749995.21** (±49959.29)	7813442.54 (±51188.79)	1344479956.2 (±82877522.43)	1439408746.27 (± 86229822.67)	1132273145.16 (± 79965554.44)	**1099654768.54** (± 80196565.13)
u_c_hilo	156633.89 (±4460.66)	159126.59 (±4204.54)	**155444.94** (± 4019.85)	156866.77 (± 4075.33)	32006013.88 (±174467.75)	31436245.11 (±168534.30)	**27991166.76** (± 103475.93)	29334134.99 (±194722.12)
u_c_lohi	250794.47 (±1106.95)	259789.25 (± 1358.68)	**248938.77** (± 1139.22)	249486.05 (± 1642.86)	40659253.17 (± 189334.26)	40815783.66 (± 195666.22)	**38932135.96** (± 125686.93)	38754185.73 (±124496.84)
u_c_lolo	5368.23 (± 54.93)	5415.26 (± 38.55)	**5298.72** (± 29.36)	5317.48 (± 34.65)	998675.05 (± 6212.75)	1016783.35 (± 6408.22)	**986519.37** (± 5597.03)	990685.08 (± 5701.38)
u_i_hihi	3125943.55 (± 28059.82)	3264988.24 (± 21965.22)	**3101124.06** (±23699.28)	3127387.94 (±29886.09)	411029656.04 (±5538259.02)	430553332.17 (±6836991.35)	**393548307.09** (±4328338.39)	402637945.53 (± 4877369.44)
u_i_hilo	76496.2 (±443.2)	77332.28 (± 356.7)	**76329.99** (± 102.3)	76489.77 (± 185.30)	15275405.57 (± 710322.2)	15706639.04 (± 712254.9)	**13085472.76** (± 621783.1)	13594467.98 (±643822.8)
u_i_lohi	112698.01 (±4438.27)	113897.98 (± 3218.33)	**109305.88** (±2846.93)	111365.16 (± 4182.55)	13099985.44 (± 753783.02)	13987925.69 (± 893540.2)	12993768.43 (± 774822.4)	**12978846.05** (±832506.3)
u_i_lolo	2658.08 (±99.38)	2648.12 (± 98.57)	2640.46 (± 89.14)	**2635.03** (± 99.12)	459989.78 (± 2551.6)	463542.49 (± 2967.7)	**454211.09** (±2018.3)	454364.32 (±1998.2)
u_s_hihi	4525479.9 (± 75342.7)	4577489.21 (± 87643.9)	**4436466.02** (± 63983.3)	4455342.87 (±79884.3)	547328532.07 (±6325448.5)	551321428.27 (±7134382.7)	**530285521.24** (± 6439958.3)	541329629.42 (±6545763.2)
u_s_hilo	99254.11 (± 322.55)	99287.03 (± 465.03)	**98784.92** (± 302.66)	98888.64 (± 205.98)	17995873.7 (±23746.2)	18569843.8 (±29668.8)	**1702866.31** (± 19345.0)	1728855.16 (±24461.3)
u_s_lohi	133262.05 (±3376.1)	135328.54 (±3865.3)	**129904.69** (±2937.6)	130627.04 (± 2833.0)	15954390.7 (± 428871.9)	15789537.4 (± 427455.3)	**15524619.07** (± 412788.1)	15652753.4 (± 423369.6)
u_s_lolo	3601.58 (±84.1)	3599.22 (±93.2)	3574.44 (± 77.8)	**3570.28** (± 69.3)	613881.24 (±6342.17)	617355.83 (±7322.02)	**603874.28** (± 6036.24)	607399.13 (±6383.4)

[Azzedin and Maheswaran, 2002] F. Azzedin and M. Maheswaran: "Integrating trust into grid resource management systems", In Proceedings of International Conference on Parallel Processing (ICPP02), 47- 54, 2002.

[Braun et al., 2001] T. D.Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund: "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", *Journal of Parallel and Distributed Computing*, 61(6):810-837, 2001.

[Casanova et al., 2000] Casanova, H., Zagorodnov, D., Berman, F., and Legrand, A. 2000: "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments", In Proceedings of the 9th Heterogeneous Computing Workshop (May 01 - 01, 2000). HCW. IEEE Computer Society, Washington, DC, p. 349.

[Freund et al., 1998] R.F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J.D. Lima, F. Mirabile, L. Moore, B. Rust, and H.J. Siegel: "Scheduling resources in multiuser, heterogeneous, computing environments with SmartNet", In 7th IEEE Heterogeneous Computing Workshop (HCW '98), 184-199, 1998.

[Garg et al., 2009] S.K. Garg, R. Buyya, and H.J. Segel: "Scheduling Parallel Aplications on Utility Grids: Time and Cost Trade-off Management", *In Proc. of the 32nd ACSC*, Wellington, Australia, 2009, CRPIT, Vol. 91, Bernard Mans Ed.

[Edlefsen and Millham, 1972] L. E. Edlefsen, and C. B. Millham: "On a formulation of Discrete N-person Noo-cooperative Games", *Metrika*, Vol. 18, No. 1, 31-34, 1972.

[Kwok et al., 2007] Y.-K. Kwok, K. Hwang and Sh. Song: "Selfish Grids: Game-Theoretic Modeling and NAS/PSA Benchmark Evaluation", *IEEE Transactions on Parallel and Distributing Systems*, Vol. 18 No. 5, 1-16, 2007.

[Pavlidis et al., 2005] N.G. Pavlidis, K.E. Parsopoulos, M.N. Vrahatis: "Computing Nash Equilibria Through Computational Intelligence Methods", *J. of Computational and Appl. Math.*, 175: 113-136, 2005.

[Song et al., 2006] Sh. Song, K. Hwang and Y-K. Kwok: "Risk-resilient Heuristics and Genetic Algorithms for Security-Assured Grid Job Scheduling", *IEEE Transactions on Computers*, Vol. 55 No 6, 703-719, 2006.

[Xhafa et al., 2007 a)] F. Xhafa, J. Carretero and A. Abraham: "Genetic Algorithm Based Schedulers for Grid Computing Systems", *International Journal of Innovative Computing, Information and Control*, Vol. 3, No. 5, 1053-1071, 2007.

[Xhafa et al., 2007 b)] F. Xhafa, L. Barolli and A. Durresi: "Batch Mode Schedulers for Grid Systems", *International Journal of Web and Grid Services*, Vol. 3, No. 1, 19-37, 2007.