USING ARTIFICIAL NEURAL NETWORK FOR MONITORING AND SUPPORTING THE GRID SCHEDULER PERFORMANCE

Daniel Grzonka and Joanna Kołodziej
Institute of Computer Science
Faculty of Physics, Mathematics and Computer Science
Cracow University of Technology
Warszawska st 24, 31-155 Cracow, Poland
E-mail: grzonka.daniel@gmail.com,
jokolodziej@pk.edu.pl

Jie Tao
Steinbuch Center for Computing
Karlsruhe Institute of Technology
Postfach 6980, D-76128 Karlsruhe, Germany
E-mail: jie.tao@kit.edu

KEYWORDS
Computational Grid, Scheduling, Artificial Neural Network, Data Classification, Security, Genetic Algorithm.

ABSTRACT

Task scheduling and resource allocations are the key issues for computational grids. Distributed resources usually work at different autonomous domains with their own access and security policies that impact successful job executions across the domain boundaries. In this paper, we propose an Artificial Neural Network (ANN) approach for supporting the security awareness of evolutionary driven grid schedulers. Making a prior analysis of the trust levels of resources and security demand parameters of tasks, the neural network monitors the scheduling and task execution processes. In the result produce the tasks-machines mapping "suggestions", which can be then utilized by the scheduler to reduce the makespan or increase the system throughput. In this paper, we report the development of risk-resilient genetic-based schedulers and their integration with an ANN module of the HyperSim-G Grid Simulator to evaluate the proposed model under the heterogeneity and large-scale system dynamics. The simulation results showed a significant impact of the ANN support on enhancing the effectiveness of the genetic-based meta-heuristics in reducing the cost of security awareness in grid scheduling.

INTRODUCTION

Grid computing is one of the most popular combinations of traditional distributed computing and utility computing. This combination has become very effective for solving large-scale complex problems from various fields.

While the maximization of the resource utilization and profits of the resource owners are the key objectives of the grid scheduling, they may conflict with grid users' security requirements and system reliability. A major hurdle in effective job outsourcing in grid is caused by network security threats. Therefore, it is desirable to have prior knowledge about the security demands from grid jobs and the trust level assured by a resource provider at the grid cluster. An effective grid scheduler must be then security-driven and resilient in response to all scheduling and risky conditions.

The specific problem discussed in this paper is the improvement of the effectiveness of grid users' and schedulers' decisions on the low-cost resource allocations with security condition recognized as the most important factor. The security awareness of the grid schedulers in our approach is supported by an Artificial Neural Network (ANN) module which is monitoring the scheduling and task execution processes. The neural network learns patterns in input data (initial tasks and machines characteristics) and produces task-machine mapping recommendations based on the stored system information, such as resource failure rates and system input parameters. Thereafter, based on the ANN "suggestions", sub-optimal schedules are generated and used in the initialization procedures of genetic-based schedulers, which may optimize the values of the main schedulers' performance metrics (in our case makespan and flowtime). Despite the generation of the sub-optimal solution to the specified scheduling problems, the ANN module is not considered in this work in fact as another (separate) scheduler. It works in a "background" of the main process and monitors the scheduling results. However, the ANN's outputs may be accepted as the optimal results by the employed scheduler.

Our contributions are summarized as the following:

- The development of an ANN-based model for supporting decision-making activities of grid actors in "secure" scheduling.
- The development risk-resilient genetic-based schedulers.
- The integration of ANN module and risk-resilient schedulers with grid simulator for their experimental evaluation.

This work considers the independent task model, where tasks submitted by the grid users are processed in a batch mode (Xhafa et al. 2007a). To incorporate security, we modified the Expected Time to Compute (ETC) model by integrating the security requirements as additional scheduling criteria. The scheduling problem

Proceedings 28th European Conference on Modelling and Simulation ©ECMS Flaminio Squazzoni, Fabio Baronio, Claudia Archetti, Marco Castellani (Editors)
ISBN: 978-0-9564944-8-1 / ISBN: 978-0-9564944-9-8 (CD)

is defined as a bi-objective global minimization task with makespan and flowtime as the main scheduler's performance measures.

We evaluate the proposed ANN model and risk-resilient schedulers under the varying heterogeneity and large-scale system dynamics by using the *HyperSim-G Grid Simulator* (Xhafa et al. 2007b). We extended the previous version of *HyperSim-G* by an ANN module, in which the *Minimal Completion Time (MCT)* algorithm is used for the generation of sub-optimal schedules.

MODELS

In this section, we first define the main scheduling attributes that are necessary for the specification of a particular scheduling problem in CGs. Thereafter, we discuss a general model of the system for a security-aware management of tasks and resources.

Scheduling Problems' Model

There are four main scheduling attributes that must be specified to define a particular tasks-machines mapping problem, namely: (a) the environment, (b) grid architecture, (c) task processing policy, and (d) tasks' interrelations.

We consider in this work an *Independent Batch Scheduling in Hierarchical CG* problem, where it is assumed that the tasks are grouped into batches and can be executed independently in a hierarchically structured dynamic grid environment.

System Model

The architecture of a CG is usually modelled as a multi-level large-scale hierarchical structure, which is a compromise between centralized and decentralized resource and task managements. This hierarchy consists of three levels: (i) meta-scheduler, (ii) local task dispatchers and (iii) clusters.

A central meta-scheduler is the main module of the system working at the highest level. The meta-scheduler interacts with local task dispatchers (brokers) and CG users to generate optimal schedules.

The brokers collect information about the "computing capacities" of the resources supplied by the resource owners within the clusters, moderate the resources, and send all of the data to the meta-scheduler. The meta-scheduler must conceive an optimal plan of the resource allocations according to the various user requirements. Thereafter, replicas of the defined schedule are sent back to the brokers.

The role of the meta-scheduler is different when security is considered as an additional criterion within 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 system brokers analyze "reputation" indexes of the machines received from the resource managers and send proposals to the scheduler.

STATEMENT OF THE PROBLEM

We start the formalization of the security aware independent batch scheduling problem by introducing the following notation for tasks and computational resources, which will be used throughout the paper:

- n – the number of tasks in a batch;
- m – the number of machines available within the system for an execution of a given batch of tasks;
- $N = \{t_1, ..., t_n\}$ – set of tasks within a batch;
- $M = \{x_1, ..., x_m\}$ – set of available machines for the task batch;
- $N_l = \{1, ..., n\}$ – labels of tasks;
- $M_l = \{1, ..., m\}$ – labels of machines.

Characteristics of Tasks and Machines

Formally, each task j can be represented by a pair of parameters $j = (wl_j, sd_j)$, where:

- wl_j is a computational load of j expressed in Millions of Instructions Per Second (MIPS), we denote by $WL = [wl_1, ..., wl_n]$ a workload vector for all tasks in the batch;
- sd_j is a security demand parameter, which is a component of a security demand vector $SD = [sd_1, ..., sd_n]$. The major attributes affecting the security demand are data integration, task sensitivity, peer authentication, access control and task execute environment.

The workload of each of the submitted task can be estimated based on the specifications provided by the users, historical data, or it can be obtained from system predictions (Hotovy 1996).

Each machine i ($i \in M_l$) in the system is represented as a triplet $i = (cc_i, ready_i, tl_i)$, where:

- cc_i – is a computing capacity of i expressed in Millions of Instructions Per Second (MIPS), we denote by $CC = [cc_1, ..., cc_m]$ a *computing capacity vector*;
- $ready_i$ – is a ready time of i, which expresses the time needed for the reloading of the machine i after finishing the last assigned task, a *ready times vector* for all machines is denoted by $ready\_times = [ready_1, ..., ready_m]$; and
- tl_i – is a trust level parameter, which specifies how much a grid user can trust a given resource manager; the manager maintains machine i status and monitors the execution of the tasks assigned to this machine; tl_i parameter

is the component of a *trust level vector* $TL = [tl_1, ..., tl_m]$. The biggest impact on the parameter are prior task execution success rate, cumulative grid cluster utilization, capabilities of firewall, intrusion detection and intrusion response.

The detailed architectural structure of CG resources is not discussed in our approach. Therefore, it must be understood that the term "machine" in our system can be a single or multiprocessor computing unit or even refer to a local small-area network. However, we assume that: (a) a task can only be executed at one CG node in each batch; (b) no preemptive process is allowed within tasks or resources; (c) when a machine fails, tasks will be reallocated to other machine(s) in the next batch; (d) when a machine processes tasks, there is no priority distinctions between the tasks assigned in the previous batches and those assigned in the current batch; and (e) a machine must remain idle when tasks have been assigned to it.

We base our approach on the fuzzy-logic trust model (developed by Song et al. 2005). In this model the task security demand is supplied by the user programs as a single parameter. The demand may appear as request for authentication, data encryption, access control, etc.

The values of the sd_j and tl_i parameters are real fractions within the range [0, 1] with 0 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.

Let us denote P_f to be a *Machine Failure Probability* matrix, the elements of which are interpreted as the probabilities of failures of the machines during the tasks executions due the high security restrictions. These probabilities denoted by $P_f[j][i]$ are calculated by using the negative exponential distribution function as follows:

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

where α is interpreted as a failure coefficient and is a global parameter of the model.

Schedule Encoding

We use in our approach two different encoding methods of schedules, which can be defined as follows.

Definition: *Let us denote by* $\boldsymbol{S}$ *the set of all permutations with repetition of the length n over the set of machine labels* M_l. *An element* $s \in \boldsymbol{S}$ *is termed a schedule and it is encoded by the following vector:*

$$s = [i_1, ..., i_n]^T, \qquad (1)$$

where $i_j \in M_l$ *denotes the number of machine on which the task labelled by j is executed. The cardinality of* $\boldsymbol{S}$ *is* m^n.

This encoding method is called a *direct representation* of the schedule.

The direct representation of the schedules can be easily transformed into a *permutation-based representation*, in which, for each machine, a sequence of tasks assigned to that machine is specified. The tasks in the sequence are sorted (in increasing order) with respect to their completion times. Thereafter, all of the task sequences are concatenated into a vector u, which is in fact a permutation without repetition of tasks to machines. Formally, this kind of schedule encoding method can be defined in the following way:

Definition: *Let us denote by* $\boldsymbol{S}_{(l)}$ *the set of all permutations without repetitions of the length n over the set of task labels* N_l. *A permutation* $u \in \boldsymbol{S}_{(l)}$ *is called a permutation-based representation of a schedule in CG and can be defined by the following vector:*

$$u = [u_1, ..., u_n]^T,$$

where $u_i \in N_l$, $i = 1, ..., n$. *The cardinality of* $\boldsymbol{S}_{(l)}$ *is n!.*

In this representation some additional information about the numbers of tasks assigned to each machine is required. Therefore, we defined a vector $v = [v_1, ..., v_m]^T$ of the size m, where v_i denotes the number of tasks assigned to the machine i.

Optimization Criteria and Objective Function

For estimating the execution times of tasks on machines we based our idea on the *Expected Time to Compute (ETC)* matrix model (Ali et al. 2000). The main structure in this model is the *ETC* matrix with estimated time needed for the completion of the task j on machine i.

In the simplest case, the entries of the $ETC[j][i]$ parameters can be computed as the ratio of the coordinates of WL and CC vectors. That is to say:

$$ETC[j][i] = \frac{wl_j}{cc_i}$$

The problem of scheduling tasks in can be measured under several criteria. In this work we consider the scheduling in CGs as a bi-objective global optimization problem with the hierarchical procedure of the minimization of *makespan* and *flowtime* objectives with makespan as a privileged criterion.

Using the *ETC* matrix model we can express the makespan and flowtime in terms of the completion times of the machines. The time of finishing the last task can be interpreted as the maximal completion time of the machines. Let us denote by *completion* a vector of the size *m*, which indicates the time that machine *i* finalizes the processing of the previously assigned and planned tasks. That is to say:

$$completion[i] = ready_i + \sum_{\substack{j\in N_l:\\ s[j]=i}} ETC[j][i] \qquad (2)$$

where s $\in$ *Schedules* denotes the schedule vector defined by Eq. (1).

The makespan can be now expressed as:

$$makespan = \max_{i\in M_l} completion[i] \qquad (3)$$

We calculate the flowtime of the sequence of tasks on a given machine *i* by using the following formula:

$$flowtime[i] = ready_i + \sum_{\substack{j\in Sort[i]:\\ s[j]=i}} ETC[j][i] \qquad (4)$$

where *Sort[i]* denotes the set of tasks assigned to the machine *i* sorted in ascending order according to their *ETC* values.

Both makespan and flowtime are expressed in arbitrary time units. In fact, the numerical values are in incomparable ranges: flowtime has a higher magnitude order over makespan and its values increase as more jobs and machines are considered. Therefore, in this approach we use *mean_flowtime = flowtime/m* for the evaluation of the flowtime criterion.

Based on the equations (2), (3) and (4) the two-steps optimization procedure can be defined as follows:

- **step 1**: minimize the maximal completion time of machines, i.e.

$$makespan = \max_{i\in M_l} completion[i] \rightarrow makespan_{min}$$

- **step 2**: minimize the flowtime without increasing the optimal makespan value, i.e.

$$current\_makespan \le makespan_{min}$$

Secure Mode Scheduling Scenario

In this paper we consider secure mode approach – when scheduler analyzes the *Machine Failure Probability* matrix in order to minimize the failure probabilities for task-machine pairs. In this scenario all of the security and resource reliability conditions are verified for the task-machine pairs. The main aim of the meta-scheduler is to design an optimal schedule for which, beyond the makespan and flowtime, the probabilities of failures of the machines during the tasks execution will be minimal. We assume that additional “cost” of the verification of security assurance condition for a given task-machine pair: (a) may delay the predicted execution time of the task on the machine and (b) is proportional to the probability of failure of the machine during the task execution. We define this “cost” as a product $P_f[j][i]\cdot ETC[j][i]$ and the completion time of the machine *i* can be calculated as follows:

$$completion[i] = ready\_time[i] + \sum_{\{j\in Tasks_i\}} \left(1 + P_f[j][i]\right) ETC[j][i]$$

where $Tasks_i$ denotes a set of tasks assigned to the machine *i* in a given batch.

ARTIFICIAL NEURAL NETWORK MODULE

For an ANN implementation we first provide a prior classification of tasks and machines available in the system based on the values of the *WL*, *CC*, *TL* and *SD* vectors. Machines are classified by the processing power (R_r classes: slowest, slower, ⋯, medium, ⋯, fastest) and the trust level (R_s number of classes: secure, less-secure, ⋯, medium, ⋯, fully-risky), where R_r and R_s are the parameters of the simulator. After the initial classification, the resources are divided into $R = R_r \cdot R_s$ classes (slowest-secure, ⋯, medium-secure, ⋯, fastest-fully-risky). We perform similar classification for the submitted tasks with workload and security demand instead of processing power and security criteria. We divide tasks into $T = T_w \cdot T_{sd}$ classes, where T_w is number of workload classes and T_{sd} is number of security demand classes. *R* machine classes and *T* task classes give us $R + T$ possible inputs for neural network.

To classify the output we need the results of monitoring the machine failures and successful execution of tasks on machines during the execution of the schedulers. We consider just the tasks which are successfully executed on the machines and then, for each machine class *k* we select the unique *major class* of tasks $\widehat{t(k)}$, which contains the greatest number of successfully executed tasks.

The network is trained by the *back propagation algorithm* (Haykin 1999) and the generated outputs are used to compose the suboptimal schedules, which are moved to the initial population of the GA-based scheduler. We implemented the *Minimum Completion Time (MCT)* algorithm for generating those suboptimal schedules with a minimal completion time. The general framework of MCT is presented in Alg. 1.

Algorithm 1 *MCT* algorithm template.

```
1: Calculate the ready_times of the machines;
2: for all Tasks in a given batch do
3:     Calculate the completion times of the machines for the tasks;
4:     Find the machine that gives minimum completion time, $m_{best}$;
5:     Assign task to $m_{best}$ machine;
6:     Update the machine completion time;
7: end for
8: return The resulting schedule;
```

SECURITY AWARE GENETIC-BASED BATCH SCHEDULERS

Heuristic methods are well known from their robustness and have been applied successfully to solve scheduling problems and general combinatorial optimization problems in dynamic environments (Aguirre and Tanaka 2007, Abraham et al. 2000, Kołodziej and Xhafa 2011). Therefore they can be considered as good candidates to be the effective CG schedulers that tackle the various scheduling attributes and additional security aspects.

In this work, we consider four genetic-based risk resilient grid schedulers presented in Table 1 that work in *secure* scheduling scenario.

Table 1: Four GA-based grid schedulers being evaluated

Scheduler	Replacement method	Scheduling scenario
GA-SS-S	Steady State	Secure
GA-SS-ANN	Steady State	Secure with ANN
GA-ST-S	Struggle	Secure
GA-ST-ANN	Struggle	Secure with ANN

Algorithm 2 A template of the genetic engine for four genetic-based grid schedulers.

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

We apply the *direct representation* of the schedules in the base populations P^t and P^{t+1}, and *permutation representation* in P^t_c and P^t_m populations. In the algorithms the initial population is generated by using the *MTC + LJFR-SJFR* method, in which all but two individuals are generated randomly. Those two individuals are created by using the *Longest Job to Fastest Resource – Shortest Job to Fastest Resource (LJFR-SJFR)* and *Minimum Completion Time (MCT)* heuristics (Xhafa et al. 2007b). In *GA-SS-ANN* and *GA-ST-ANN* algorithms the initial populations contain additionally the schedule generated by ANN module (by using the MCT heuristics).

The aforementioned methodologies differ in the implementation of the replacement mechanism in the main genetic framework. We used *Steady State* replacement in *GA-SS-xxx* algorithms and *Struggle* procedure in *GA-ST-xxx*.

In the *Steady State* method, a set of the best offspring replaces the worst solutions in the old base population. The main drawback of this method is that it can lead to premature convergence on some solution and impacts on the stagnation of the population. However, the aforementioned may be very fast in the fitness reduction.

A *Struggle* replacement mechanism can be an effective tool for avoiding too fast scheduler's convergence to the local optima. In this method, new generation of individuals is created by replacing a part of the population by the most similar individuals – if this replacement minimizes the fitness value.

The *Struggle* strategy has shown to be very effective in solving several large-scale multiobjective problems (see e.g., Bartschi Wall 1996, Grueninger 1997). However, the computational cost can be very high, because of the need of calculation of distances among all offspring in resulting population and the individuals in the base population for the current GA loop. To reduce the execution time of the struggle procedure we use a *hash technique*, in which the hash table with the *task-resource allocation* key is created. The value of this key, denoted by K, is calculated as the sum of the absolute values of the subtraction of each position and its precedent in the direct representation of the schedule vector (reading the schedule vector in a circular way). The hash function f_{hash} is defined as follows:

$$f_{hash}(K) = \begin{cases} 0, & K < K_{min} \\ \left\lfloor N \cdot \left(\frac{K - K_{min}}{K_{max} - K_{min}}\right) \right\rfloor, & K_{min} \le K < K_{max} \\ N - 1, & K \ge K_{max} \end{cases}$$

where K_{min} and K_{max} correspond respectively to the smallest and the largest value of K in the population, and N is the population size.

EXPERIMENTAL EVALUATION

In this section we present the results of the experimental evaluation of four genetic-based schedulers defined in Table 1 in dynamic grid environment by using the grid simulator. The main aim of our simple analysis is to compare the effectiveness of the proposed metaheuristics in scheduling scenario and to verify the impact of the activation of ANN module on the failures of the machines as well as on the makespan and flowtime optimization.

Security Aware HyperSim-G Grid Simulator

To simulate the secure scheduling we define a *Secure HyperSim-G* simulator by extending the HyperSim-G framework. HyperSim-G simulator is based on a discrete event model. The sequence of events and the changes in the state of the system capture the realistic grid dynamics. The simulator provides the full simulation trace by indicating a parameter for the trace generation. This functionality is useful for an easy implementation of the Neural Network module.

Based on the ETC Matrix model the Secure HyperSim-G simulator generates an instance of the scheduling problem by using the following input data: (i) the trust level vector *TL*, (ii) the security demand vector *SD*, (iii) the workload vector of tasks *WL*, (iv) the computing capacity vector of machines *CC*, (v) the vector of prior loads of machines *ready_times*, and (vi) the *ETC* matrix. The Neural Network module is designed for supporting the resolution methods used in the *Scheduler* class of the simulator. The output of the network is used to define a suboptimal schedule, which is copied to the initial population of a GA-based scheduler.

Experimental Settings and Performance Metrics

The Secure HyperSim-G simulator is highly parameterized to reflect the various realistic grid scenarios. The values of key input parameters for the simulator are presented in Table 2.

Table 2: Values of key parameters of the grid simulator

Parameter/Size	Small	Large
Hosts (init., max, min)	32, 37, 27	256, 264, 248
Resource cap. (in MHz CPU)	N(5000, 875)	
Add host	N(625000, 93750)	N(437500, 65625)
Delete host	N(625000, 93750)	
Tasks (init., total)	384, 512	3072, 4096
Inter arrival	E(7812.5)	E(976.5625)
Workload	N(250000000, 43750000)	
Security demands sd_j	U[0.6; 0.9]	
Trust levels tl_i	U[0.3; 1]	
Failure coefficient α	3	

We use the notation *U[x, y]*, *N(a, b)* and *E(c, d)* for uniform, Gaussian and exponential probability distributions respectively.

For activating the ANN module we divided the tasks and machines into 18 classes: 9 for processing power and trust level criteria (machines) and 9 for workload and security demand criteria (tasks). The ANN contains two hidden layers, the weight coefficients are in the range of [-0.2; 0.2] and the learning rate is 0.01. The training set for ANN contains the characteristics of the tasks and machines and the task-machine matching results collected after the 500 runs of the simulator with inactive Neural Network module.

The key parameters for all types of genetic-based schedulers are presented in Table 3.

Table 3: GA-based schedulers settings

Parameter	Value
Evolution steps	5*n
Population size	60
Intermediate pop.	48
Cross probab.	0.9
Mutation probab.	0.15
Max time to spend	400 sec

We used the following three metrics to evaluate the scheduling performance:

- *Makespan*,
- *Flowtime*, and
- *FailureRate* F_r parameter defined as follows:

$$F_r = \frac{n_{failed}}{n} \cdot 100\%$$

where n_{failed} denotes the number of unfinished tasks, which must be rescheduled.
Each experiment was repeated 30 times under the same configuration of operators and parameters.

Results

In Fig. 1 and 2 we present the results of our experiments for four genetic-based risk resilient Schedulers.

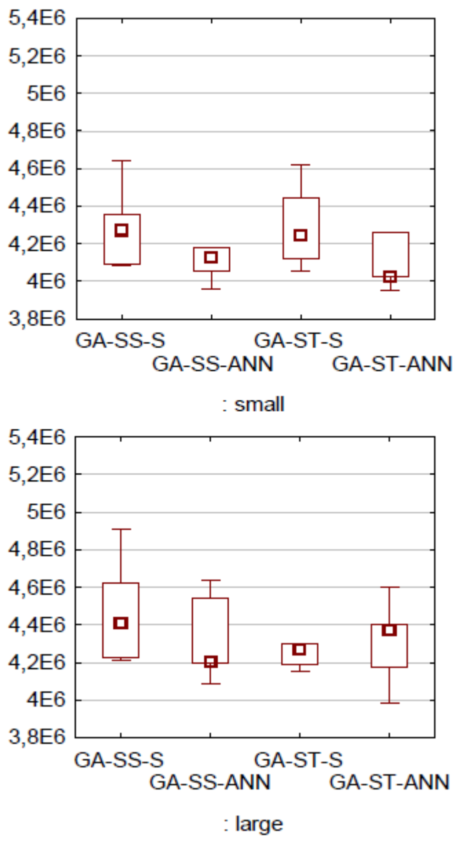

Fig. 1. The box-plot of the results for makespan.

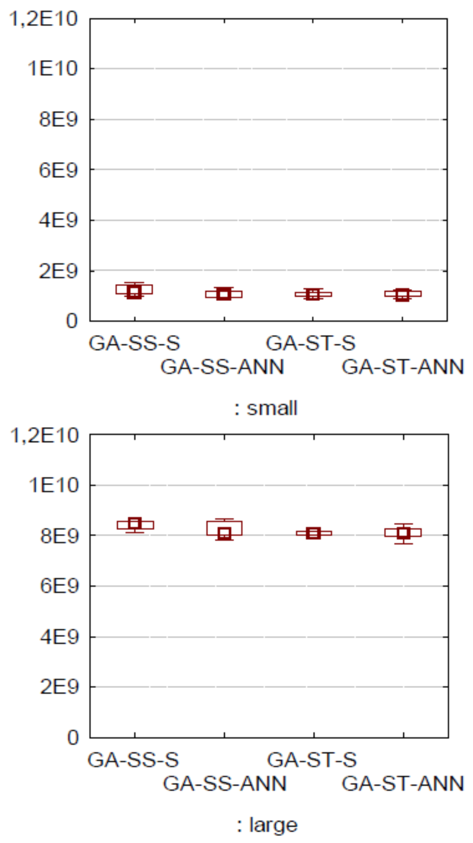

Fig. 2. The box-plot of the results for flowtime.

The best results in the makespan optimization have been achieved by both *GA-XX-ANN* schedulers. The efficiency of the ANN support can be observed especially in the 'Large' grid scenario. While in the small grid case the differences in the averaged makespan values are not so big, in second scenario *GA-SS-S* and *GA-ST-S* meta-heuristics significantly lag behind the secure schedulers. It can be also observed that in all instances the distribution of the results is asymmetric.

In the case of the flowtime minimization both *GA-XX-ANN* meta-heuristics outperform again the rest of the methods, however the differences in the results are not as significant as it was in the makespan case. Additionally, we noted that as the instance size is doubled, the flowtime values increase considerably for all applied schedulers, while the makespan is almost at the same level. Another observation is that all schedulers are rather 'symmetric' in sense of distribution of the results. The best relative effectiveness of the ANN support may be observed in 'Large' grid cases.

As the stopping criterion we consider in this work the maximal and a priori defined number of generations for which the GAs are activated. However, the solutions generated by ANN may not be improved by the GA metaheuristics, and the search process can be stopped because of the stagnation in the improvement of the solutions' qualities. Therefore we additionally 'measured' for each scheduler the minimal time (expressed in the genetic epochs (generations)) necessary for the generation of the best solutions found by such a scheduler. The results are presented in Table 4. In parentheses we displayed the relative effectiveness parameter *ef*, which is calculated as the ratio of the minimal number of genetic epochs necessary for finding the optimal solutions and the stopping criterion, which is $5 \cdot n$, where n denotes the number of tasks in the system.

It can be noted than ANN module in most of the instances reduced the time necessary for finding the best possible solutions by each scheduler, so it may be also helpful in optimizing the whole process by stopping the algorithms much sooner that it was originally set, in most of the cases the time may be reduced approximately by 30–40 %.

Table 4: The number of genetic epochs necessary for the generation of the best solutions (efficiency parameter *ef* [%])

Strategy/Size	Small	Large
GA-SS-S	1831 (71.52%)	19002 (92.78%)
GA-SS-ANN	1622 (63.60%)	**17830 (87.06%)**
GA-ST-S	1703 (68.52%)	17993 (88.63%)
GA-ST-ANN	**1511 (60.44%)**	17910 (87.83%)

The effectiveness of the ANN support is confirmed by the lowest failure rates achieved by the *GA-XX-ANN* schedulers. The results for all four schedulers are presented in Table 5. In all instances the schedulers with the active ANN module outperform the other methods. The suboptimal solutions produced by ANN allow reducing the machine failures by 1% – 6%.

Table 5: Average values of Failure Rate parameter

Strategy/Size	Small	Large
GA-SS-S	6.104	8.943
GA-SS-ANN	4.88	**5.026**
GA-ST-S	9.218	5.744
GA-ST-ANN	**4.093**	7.894

CONCLUSIONS

In this paper we present the implementation of the Artificial Neural Network (ANN) as the support mechanism for risk resilient genetic-based schedulers in computational grids. Making a prior analysis of trust levels of the resources and security demand parameters of tasks, the neural network monitors the scheduling and task execution processes. The network learns patterns in input (initial tasks and machines characteristics) and produce the tasks-machines mapping suggestions as the outputs based on the stored data, which includes information about the resource failures. Then, based on the ANN 'suggestions' sub-optimal schedules are generated, which are then used to modify the initialization procedures of genetic scheduling algorithms.

We have evaluated the proposed model under the heterogeneity, scalability and dynamics conditions using the Secure HyperSim-G Grid Simulator. We integrated a *Neural Network* module with the simulator, where *Minimal Completion Time (MCT)* algorithm is used for the sub-optimal schedules generation. The relative performance of four variants of the GA-based schedulers was measured for the makespan, flowtime and machines' failure rates metrics in secure scenario. We have demonstrated the efficiency of the schedulers supported by the ANN module in a fast reduction of the makespan and flowtime values and the improvement of the reliability of the resources. The best effectiveness of the ANN support can be observed in the makespan minimization in all considered grid scenarios. The obtained simulation results suggest that it is more resilient in the grid environment to pay some additional scheduling 'cost' due to verification of the security conditions instead of taking a risk on unreliable resources allocated.

Worth considering the extension of the comparison analysis to a wider set of meta-heuristic schedulers and introduce the multi-class tasks classification for generating the ANN outputs. It also will be interesting to provide some experimental study on the online scheduling, where some online learning mechanisms, like neuro-fuzzy systems (Nauck 1997) can be implemented to improve the scheduling results.

REFERENCES

Abraham, A.; R. Buyya and B. Nath. 2000. "Natures heuristics for scheduling jobs on computational grids", *Proceedings of the 8th IEEE ACC*, India.

Aguirre H.E., and K. Tanaka. 2007. "Working principles, behavior, and performance of MOEAs on MNK-landscapes", *European Journal of Operational Research*, vol. 181, 1670-1690.

Ali, S.; H.J. Siegel; M. Maheswaran and D. Hensgen. 2000. "Task execution time modelling for heterogeneous computing systems", *Proceedings of Heterogeneous Computing Workshop*, 185–199.

Bartschi Wall, M. 1996. "A Genetic Algorithm for Resource-Constrained Scheduling", *PhD Thesis*, Massachusetts Institute of Technology, MA.

Garg, S.K.; R. Buyya and H.J. Segel. 2009. "Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-off Management", *In Proc. of the 32nd ACSC*, Wellington, New Zealand. CRPIT, 91. Mans, B., Ed. ACS., 139–147.

Grueninger, T. 1997. "Multimodal optimization using genetic algorithms", *Technical report*, Department of Mechanical Engineering, MIT, Cambridge, MA.

Haykin, S. 1999. *Neural Networks: A Comprehensive Foundation*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ.

Hotovy, S. 1996. "Workload evolution on the Cornell Theory Center IBM SP2", in *Job Scheduling Strategies for Parallel* Proc. Workshop, IPPS'96, 27–40.

Kołodziej, J. and F. Xhafa. 2011. "Enhancing the genetic-based scheduling in computational Grids by a structured hierarchical population", *Future Generation Computer Systems*, vol. 27, DOI:10.1016/j.future.2011.04.011, 1035-1046.

Nauck, D.; F. Klawonn and R. Kruse. 1997. *Neuro-Fuzzy Systems*, John Wiley & Sons.

Song, S.; K. Hwang and Y.K. Kwok. 2005. "Trusted Grid Computing with Security Binding and Trust Integration", Journal of Grid Computing.

Xhafa, F.; L. Barolli and A. Durresi. 2007. "Batch Mode Schedulers for Grid Systems", *International Journal of Web and Grid Services* 3(1), 19–37.

Xhafa, F.; J. Carretero and A. Abraham. 2007. "Genetic Algorithm Based Schedulers for Grid Computing Systems", *International Journal of Innovative Computing, Information and Control*, vol. 3, No. 5, 1053-1071.

AUTHOR BIOGRAPHIES

DANIEL GRZONKA received his B.Sc. and M.Sc. degrees with distinctions in Computer Science at Cracow University of Technology, Poland, in 2012 and 2013, respectively. Actually, he is Ph.D. student at Jagiellonian University in cooperation with Polish Academy of Sciences. He is also a member of Polish Information Processing Society. His e-mail address is: grzonka.daniel@gmail.com.

JOANNA KOŁODZIEJ graduated in Mathematics from the Jagiellonian University in Cracow in 1992, where she also received the PhD in Computer Science in 2004. She works as an assistant professor at Cracow University of Technology. She has served and is currently serving as PC Co-Chair, General Co-Chair and IPC member of several international conferences and workshops including PPSN 2010, ECMS 2011, CISIS 2011, 3PGCIC 2011, CISSE 2006, CEC 2008, IACS 2008-2009, ICAART 2009-2010. Dr Koodziej is a EB member and guest editor of several peer-reviewed international journals and author and co-author of many publications in high quality peer reviewed international journals. Her e-mail address is: jokolodziej@pk.edu.pl and her Web-page can be found at http://www.joannakolodziej.org.

JIE TAO got her PhD degree at the department of Computer Science of Munich University of Technology, Germany. She is currently a senior research associate at the Steinbuch Centre for Computing, Karlsruhe Institute of Technology. Dr. Tao's research work targets mainly on parallel and distributed computing, data-intensive computing as well as Grid and Cloud computing. Dr. Tao has published a number of articles in peer-reviewed international journals and leading conferences. She serves as co-chair or PC member in a set of international conferences and workshops. She is a guest editor of several international journals. Her e-mail address is: jie.tao@kit.edu.