Artificial Neural Network support to monitoring of the evolutionary driven security aware scheduling in computational distributed environments

Daniel Grzonka a, *, Joanna Kołodziej a, Jie Tao b, Samee Ullah Khan c

a Cracow University of Technology, Institute of Computer Science, ul. Warszawska 24, 31-155 Cracow, Poland
b Steinbuch Center for Computing, Karlsruhe Institute of Technology, Postfach 6980, D-76128 Karlsruhe, Germany
c North Dakota State University, Fargo, ND 58108, USA

HIGHLIGHTS

• We have developed ANN-based model for monitoring and supporting the grid schedulers.
• We have developed six security-aware genetic schedulers.
• Our model was evaluated under the heterogeneity and large-scale simulated system.
• Schedulers supported by ANN achieved better results.
• The number of the genetic epochs has been reduced.

ABSTRACT

Monitoring of the system performance in highly distributed computing environments is a wide research area. In cloud and grid computing, it is usually restricted to the utilization and reliability of the resources. However, in today’s Computational Grids (CGs) and Clouds (CCs), the end users may define the special personal requirements and preferences in the resource and service selection, service functionality and data access. Such requirements may refer to the special individual security conditions for the protection of the data and application codes. Therefore, solving the scheduling problems in modern distributed environments remains still challenging for most of the well known schedulers, and the general functionality of the monitoring systems must be improved to make them efficient as schedulers supporting modules.

In this paper, we define a novel model of security-driven grid schedulers supported by an Artificial Neural Network (ANN). ANN module monitors the schedule executions and learns about secure task–machine mappings from the observed machine failures. Then, the metaheuristic grid schedulers (in our case—genetic-based schedulers) are supported by the ANN module through the integration of the sub-optimal schedules generated by the neural network, with the genetic populations of the schedules.

The influence of the ANN support on the general schedulers’ performance is examined in the experiments conducted for four types of the grid networks (small, medium, large and very large grids), two security scheduling modes—risky and secure scenarios, and six genetic-based grid schedulers. The generated empirical results show the high effectiveness of such monitoring support in reducing the values of the major scheduling criteria (makespan and flowtime), the run times of the schedulers and the grid resource failures.

1. Introduction

In the recent years, distributed computing became a common paradigm that has been widely used in solving the large scale complex optimization problems in science, social modeling, engineering, medicine and finance [1–3]. One of the most popular
models of computational distributed environments are grid and cloud systems. Computational grids (CGs) are primarily concerned with the development of high-performance applications, which can be executed simultaneously on multiple computers or supercomputers connected by the wide area networks. Computational Clouds (CCs) can be defined in general as a compilation of the large networks of virtualized services of various types of resources, namely hardware resources (CPU, storage, and network) and software resources (e.g., databases, message-queueing systems, monitoring systems, load-balancers). In industry these services are referred to as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Cloud computing services are hosted in large data centers, often referred to as data farms. Cloud computing gives application developers the ability of the access to the virtual infinite resources with ‘pay-per-use’ and ‘as needed’ utilization options. Along with the development of the distributed computing environments, the role of the monitoring systems of such environments also changed, especially in the Big Data era. Currently it is expected that monitoring systems’ functions are not just restricted to the collection of the information about the system state and condition, but also may be helpful in the automation of the process of benchmark and testbed generation and may support the decision processes.

In our work we focused on computational grids, which can be considered as an infrastructure layer of CCs [4]. In CGs, an optimal task scheduling and low cost resource allocation are crucial in the general infrastructure utilization and improvement of the system performance. In large-scale CGs, distributed resource clusters usually work at different autonomous domains with their own access policies that impact the successful execution of the submitted tasks across the domain boundaries. The problem becomes more complex if we take into account the individual needs of the grid end users, mainly because of the security policy. A major hurdle in effective task outsourcing in grid is caused by network security threats. The grid resources may not be accessible if the grid cluster is under attack. The system infections may lead to machine crashes during the execution of tasks dispatched to that cluster. Therefore, it is desirable to possess a deep knowledge about tasks’ security demands and resources’ trust levels. 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 by a grid site or by a cluster node (security) and the behavior of a resource node (trust) must be defined and analyzed.

There have been a number of studies exploiting the security aspects in grid scheduling in risky environments, where the resource trust parameters must be analyzed. Song et al. [5,6] developed a security aware model in online grid scheduling, where security demand and trust level are expressed as scalar parameters. Resource reliability and security have been defined as additional scheduling criteria in independent grid scheduling [7]. Matching grid users’ security requirements and the ‘reputation’ of the grid clusters impact the behavior and strategies of users, task managers, and resource brokers. A game-theoretical support to the users’ decision making activities was presented in our previous work [8]. All these works are of great importance and relevance to our current study on the security grid scheduling. However, analyzing the security assurance conditions for each task–machine pair in the case of online scheduling or solving the users’ games is a complex and time consuming process. The aforementioned also increases the complexity of the scheduling algorithms and cannot prevent the resource failures during the tasks execution. The role of the monitoring grid module, usually restricted just for the notification of the resource failures during the execution of the generated schedules, may be therefore extended to the comprehensive analysis of the monitoring results and the system condition. Such knowledge may be used for the automatization of the generation of the testbeds in the wide area networks [9], but also for supporting the grid schedulers. In this last approach, the system should be able to learn from the monitoring results in order to define some ‘suggestions’ for the scheduler about the improvement of the task–machine mappings.

In this paper, we developed an Artificial Neural Network (ANN) grid monitoring model, which supports the work of the grid schedulers in the case of special personal end users’ requirements on the data and application codes protection. Making a prior analysis of trust levels of the resources and security demand parameters of tasks, the neural network is monitoring the scheduling and task execution processes. The ANN module works in the background of the schedulers in the grid system with periodical synchronization with both scheduling and task execution modules. ANN learns initial tasks and machines characteristics and generates the task–machine mapping recommendations based on that characteristics and results of the monitoring process. Such ANN ‘suggestions’ are defined as additional sub-optimal schedules and are sent to the genetic populations of the other schedules evaluated and processed by the genetic schedulers. It is expected that such support may improve the performance and effectiveness of the schedulers by reduction of the machine failure rates. In fact, our technique leads to a simple categorization of the resources available within the system from risky to the most trustworthy machines, which have a great impact on the ‘reputation’ indexes of the grid clusters. Despite the generation of the sub-optimal solution to the specified scheduling problems, the ANN module is not considered in this work as another (additional) scheduler. However, the ANN’s outputs may be accepted as the sub-optimal results in the case of stagnation in the improvement of the solutions’ qualities by the active scheduler.

Our contributions are summarized as follows:

• The development of the complete ANN-based model for supporting decision-making activities of grid actors in ‘secure’ scheduling.

• The integration of ANN module and risk-resilient schedulers with the developed security-aware grid simulator for their experimental evaluation in four grid architectures.

In order to address the security issues in scheduling model, we modified the Expected Time to Compute (ETC) model [10] by integrating the security requirements as additional scheduling criteria. The scheduling problem is defined as a bi-objective global minimization task with makespan and flowtime as the main scheduler’s performance measures.

The influence of the ANN support on the results of the grid scheduling has been verified experimentally. Such empirical analysis has been provided according to the following scheduling objectives: (i) makespan and flowtime optimization, (ii) the verification of the reliability of resources, and (iii) minimization of the schedulers’ run times (see Section 6). We developed six genetic-based (GA-based) grid scheduling strategies with the Steady State or Struggle replacement methods implemented in the risky, secure or secure supported by ANN module scheduling scenarios. The GA-based schedulers work in static and dynamic grid environments with 32, 64, 128 and 256 hosts.

We have developed for the experiments a secure grid simulator – Secure HyperSim-G – which is the extension of the HyperSim-G Grid software [11]. Such extension includes the implementation of the ANN module, additional security-related task and machine characteristics generator and specification of the risky and secure versions of the GA schedulers. Our modification of the simulator allows the flexible activation or inactivation of all of the scheduling criteria and modules. The aforementioned makes the simulator well adapted to the proper illustration of the different realistic scenarios and reduces the possible restriction to the specific
scheduling resolution methods (in our case just population-based meta-heuristics).

The research presented in this paper is an extended version of our preliminary results published in [12] and presented at the 28th European Conference on Modelling and Simulation (ECMS2014) in Brescia (Italy). Compare to our preliminary work, we extended the model specification and integration of the ANN module with the grid simulator. The coherence of the ANN work and the scheduler and simulator works has been improved by the extension of the period of the neural training set generation. We have also defined new grid scenarios (large and very large), introduced dynamic and static scheduling modes for the grid system, and improved the ETC-based models in secure and risky scheduling scenarios.

The rest of the paper is organized as follows. Section 2 describes the secure grid architectural model and the scheduling attributes considered in this paper. In Section 3 we briefly define the independent job scheduling problem and two scheduling scenarios, namely risky and secure modes, which lead to the formulation of the main scheduling objectives. We present the ANN module and the general framework of risk-resilient genetic-based schedulers in Sections 4 and 5. Section 6 describes experimental settings for the simulator and schedulers and performance metrics used in our evaluation study followed by experimental results. Related work is discussed in Section 7. Then we summarize our work and draw simple conclusions in Section 8.

2. Models

Unlike classical distributed systems, where all system entities work under the same policy, in CGs the users, managers, and resource providers may operate in different autonomous domains with incoherent local policies. Therefore scheduling in grids can be considered as a family of highly parametrized problems, which makes mapping jobs to machines a challenging NP-complete global optimization task.

2.1. Scheduling problems in CGs

Different types of scheduling problems in CGs may be distinguished with respect to different properties of the underlying grid environment and various requirements of the users. To achieve the desired performance of the system, both users’ conditions and grid environment information must be “embedded” into the scheduling mechanism.

We present in Fig. 1 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 static or dynamic grid environments. Due to the massive capacity of parallel computation in CGs, this kind of scheduling is very useful in illustrating a lot of realistic scenarios, where the users, independent of each other, submit their jobs to the system, all grid-enabled applications run periodically, and large amount of data are simultaneously transferred, replicated, and accessed by those applications. Real life examples of batch scheduling include: (a) processing of large log data files of online systems (e.g. banking systems, virtual campuses, and health systems), (b) processing of large data sets from scientific experimental simulations (e.g. High Energy Physics and Parameter Sweep Applications), and (c) data mining in bio-informatics applications. All needed system/application attributes are marked as dark-background components of the diagram in Fig. 1.

2.2. System model

Considered a multi-level large-scale hierarchical architecture of a CG is a compromise between centralized and decentralized resource and task managements. Generally this hierarchy consists of two or three levels as it is demonstrated in Fig. 2.

The main module of the system – central meta-scheduler – is located at the highest level of the grid multi-layer scheduling model. This meta-scheduler is responsible for generating optimal schedules based on users requirements and information collected by local task dispatchers (brokers).

The brokers (also called cluster managers) mediate between meta-scheduler and infrastructure and collect information about the ‘computing capacities’ of the resources. The knowledge of the system, exploration and exploitation by the local brokers are limited to the grid clusters. Prepared data are sent to the main module. The meta-scheduler must conceive an optimal plan of the resource allocations according to the various user requirements, such as the deadline and budget constraints. Thereafter, replicas of the defined schedule are sent back to the brokers.

In the case of security-awareness considered as an extra scheduling attribute, the role of the main module is different than in ‘conventional’ grid scheduling. 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. Moreover, the brokers control the resource allocation and communication between CG users and resource owners.

---

1 In today’s grid applications a region of activity of such a meta-scheduler is in fact restricted to a part of the global network of the grid clusters. That is to say that there are many fully cooperating meta-schedulers in CGs, which may be geographically distributed or belong to different companies, networks or organization with very restricted resource sharing policies and security conditions.
3. Statement of the problem

In this section, following our previous work [12], we formalize considered problem by introducing corresponding notation and characteristics of tasks and machines. Next we define our approach for schedule encoding, optimization criteria and extended scheduling scenarios.

The security aware independent batch scheduling problem is represented by the following notation:

- $n$—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;
- $N = \{t_1, \ldots, t_n\}$—set of tasks within a batch;
- $M = \{x_1, \ldots, x_m\}$—set of available machines for the task batch;
- $N_t = \{1, \ldots, n\}$—labels of tasks;
- $M_t = \{1, \ldots, m\}$—labels of machines;

In the following paragraphs we present brief characteristics of the main components of the system model.

### 3.1. Characteristics of tasks and machines

We define tasks as monolithic applications or meta-task with no dependencies among the components.

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 $WL = [wl_1, \ldots, wl_n]$ 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 $SD = [sd_1, \ldots, sd_n]$.

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 [13].

Each machine $i$, $i \in M$, 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 $CC = [cc_1, \ldots, cc_m]$ 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 $ready\_times = [ready_1, \ldots, ready_m]$; 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 $i$'s status and monitors the execution of the tasks assigned to this machine; $tl_i$ parameter is the component of a trust level vector $TL = [tl_1, \ldots, tl_m]$.

In our work we assume that: (i) a structure of machines is not considered—it can be a single or multiprocessor computing unit or even small local network; (ii) a task can only be executed at one CG node in each batch; (iii) no preemptive process is allowed within tasks or resources; (iv) when a machine fails, tasks will be reallocated to other machine(s) in the next batch; (v) 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 (vi) a machine must remain idle when tasks have been assigned to it.

The trust level and security demand parameters are expressed as scalar quantities, which are generated by the aggregation of several scheduling and system attributes at users’ and resource owners’ sites. We base our approach on the fuzzy-logic trust model developed by Song et al. [6]. 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 trust level parameters of a resource clusters are aggregated through a two-level hierarchic fuzzy-logic based trust procedure, in which: (i) in the lower intra-site level there are applied two fuzzy inference systems for the evaluation of the self-defense capabilities and trust indexes of the resources, each grid cluster reports its assessed self-defense capability to all other clusters; (ii) in the inter-site level there are collected the inputs from all resource clusters and the trust level vector is defined through another fuzzy inference process.

The major behavior and intrinsic security attributes required for the specification of $SD$ and $TL$ vectors are presented in Fig. 3 (see also [5]). All such attributes change dynamically and depend heavily on the security policy, accumulated resource or grid cluster “reputation”, self-defense capability, attack history, special users’ requirements, and peer authentication.

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.

In order to estimate the probabilities of process failures, we introduced a Machine Failure Probability (MFP) matrix. Matrix elements represent the probabilities of failures of the machines during the tasks executions due to the high security restrictions. $MFP$ matrix is calculated by using the negative exponential distribution function as follows:

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

where $\alpha$ is interpreted as a failure coefficient and is a global parameter of the model.
The process of matching $sd_i$ with $tl_i$ is similar to that of a real-life scenario where users of some portals like Yahoo! are required to specify the security level of the login session.

3.2. Schedule encoding

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

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

$$s = [i_1, \ldots, i_n]^T,$$

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

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

The set $\mathcal{S}$ can be also defined as the Cartesian product of $n$ copies of the $M$ sets. That is to say:

$$\text{Schedules} = M_1 \times \cdots \times M_n.$$  \hspace{1cm} (3)

The cardinality of $\mathcal{S}$ is $m^n$.

In our work we transformed the direct representation into a permutation-based representation, in which for each machine a sequence of sorted (in increasing order to their completion times) tasks is assigned. 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 $\mathcal{S}(1)$ the set of all permutations without repetitions of the length $n$ over the set of task labels $N$. A permutation $u \in \mathcal{S}(1)$ is called a permutation-based representation of a schedule in CG and can be defined by the following vector:

$$u = [u_1, \ldots, u_n]^T,$$

where $u_i \in N$, $i = 1, \ldots, n$. The cardinality of $\mathcal{S}(1)$ is $n!$.

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

**Example.** The following vector $[1, 2, 1, 4, 3, 1, 2, 4, 3, 3]^T$ is an example of the schedule for 4 machines and 10 tasks encoded by the direct representation method. The same schedule in the permutation-based representation is as follows: $[[1, 3, 6, 2, 7, 5, 9, 10, 4, 8], [3, 2, 3, 2]^T]$.

3.3. Scheduling scenarios and objectives

For estimating the execution times of tasks on machines we based our idea on the Expected Time to Compute (ETC) matrix model [10]. The main structure in this model is the ETC matrix, $ETC = [ETC[j][i]]_{n \times m}$, where $ETC[j][i]$ denotes the expected (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:

$$ETC[j][i] = \frac{w_i}{cc_j}.$$  \hspace{1cm} (5)

All of the values of $w_i$ and $cc_j$ are generated by using the Gamma or simple Gaussian probability distributions for the expression of tasks and machines heterogeneities in the grid system. In cases when: (a) meta-tasks are submitted by the users and (b) multi-processor machines are proposed by the resource providers, the values of $ETC$ matrix can be computed by using some special local scheduling policies and resolution methods.

3.3.1. Optimization criteria and objective function

The problem of scheduling tasks in CG is multi-objective in its general setting as the quality of the solutions 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.

**Definition.** Let us denote by $F_j$ the time of finalizing task $j$ and let $\text{Schedules}$ be a set of directly encoded schedules in a given batch (see Definition 3.2). Then:

- A makespan is defined as the finishing time of the latest task in the batch. That is to say:
  $$\text{makespan} = \min_{s \in \text{Schedules}} \max_{j \in N_s} F_j.$$  \hspace{1cm} (6)

- We define a flowtime as the sum of the finalization times of all the tasks in the batch in the following way:
  $$\text{flowtime} = \min_{s \in \text{Schedules}} \sum_{j \in N_s} F_j.$$  \hspace{1cm} (7)

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

Using the ETC matrix model we can express the makespan and flowtime in terms of the completion times 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:

$$\text{completion}[i] = \text{ready}_i + \sum_{j \in \text{Tasks}} \text{ETC}[j][i]$$

where $s \in \text{Schedules}$ denotes the schedule vector defined by Eq. (2).

The time of finishing the last task can be interpreted as the maximal completion time of the machines:

$$\text{makespan} = \max_{i \in \text{MACHINES}} \text{completion}[i]$$

(9)

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

$$\text{flowtime}[i] = \text{ready}_i + \sum_{j \in \text{Sort}[i]} \text{ETC}[j][i]$$

(10)

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

Based on Eqs. (8)-(10) the two-steps optimization procedure can be defined as follows:

- **step 1**: minimize the maximal completion time of machines, i.e.
  $$\text{makespan} = \max_{i \in \text{MACHINES}} \text{completion}[i] \rightarrow \text{makespan}_{\text{min}}$$

- **step 2**: minimize the flowtime without increasing the optimal makespan value, i.e.
  $$\text{current}_{\text{makespan}} \leq \text{makespan}_{\text{min}}$$

3.3.2. Scheduling scenarios

To extend our research we considered the situation when the grid cluster or resource may be not accessible to the global meta-scheduler. For example it can be infected with intrusions or be malicious attacks. The scheduler has two options to consider: (a) to analyze the Machine Failure Probability matrix in order to minimize the failure probabilities for task–machine pairs; or (b) to perform an “ordinary” scheduling without any prior analysis of the security conditions, abort the task scheduling in the case of machine failure, and reschedule this task at another resource. We refer the aforementioned scheduling scenarios secure and risky modes, respectively.

**Secure mode.** 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:

$$\text{completion}[i] = \text{ready}_i + \sum_{j \in \text{Tasks}} (1 + P_f[j][i] \cdot ETC[j][i])$$

(11)

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

**Risky mode.** In this scenario, all risky and failing conditions are ignored. The scheduling process is realized as a two-step procedure. First, the scheduling is performed just by analyzing the ETC matrix. If failures are observed during some tasks execution, then the unfinished tasks are temporarily moved into the backlog set. This set is defined as a considered batch supplement and the tasks are rescheduled as in the secure mode. The total completion time of machine $i (i \in \text{MACHINES})$ in this case can be defined as follows:

$$\text{completion}'[i] = \text{completion}_{\text{II}}[i] + \text{completion}_{\text{III}}[i]$$

(12)

where completion$_{\text{II}}[i]$ is the completion time of machine $i$ calculated by using Eq. (8) for tasks primarily assigned to the machine, and completion$_{\text{III}}[i]$ is the completion time of machine $i$ calculated by using Eq. (11) for rescheduled tasks, i.e. the tasks moved to the machine $i$ from the other resources.

Although the probabilities of machines' failures are expected to be higher in the risky than in the secure mode, there is certainly no guarantee of the successful execution of all tasks in the security scenario. It can be observed that if the security assurance condition is satisfied for each task–machine pair (i.e. $sd_i \leq tl_i$ for $i \in \text{MACHINES}$, $j \in \text{NAMES}$), the completion times of machines in both secure and risky modes are identical with the completion times defined for standard independent scheduling problem (see Eq. (8)), where it is assumed that each task must be successfully executed on each machine and no security requirements are analyzed.

4. Artificial Neural Network (ANN) module

The ANN module in our system is responsible for monitoring the machine failures as a consequence of too strong security conditions in the scheduling process (and non-optimal task–machine mapping). The ANN module works in the background of the scheduling process (and does not increase the schedulers' complexities!!) and, thanks to the internal learning system, may ‘suggest’ additional sub-optimal solutions for the schedulers, in order to improve their effectiveness in the reduction of the running times, values of the major scheduling criteria and machine failure rates.

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 [12]. Machines are classified by the processing power ($R_\text{I}$ classes: slowest, slower, ..., medium, ..., fastest) and the trust level ($R_\text{II}$ number of classes: secure, less-secure, ..., medium, ..., fully-risky), where $R_\text{I}$ and $R_\text{II}$ are the parameters of the simulator. After the initial classification, the resources are divided into $R_\text{I}$, $R_\text{II}$ 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 $T = T_w \cdot T_d$ classes, where $T_w$ is number of workload classes and $T_d$ is number of security demand classes. $R_\text{I}$ machine classes and $T$ task classes give us $R_\text{I} \cdot T$ possible inputs for neural network.

Formally, the input data can be represented by the following pair of vectors:

$$\left[\text{TASKS}_\text{MX}, \text{MACHINES}_\text{MX}\right]$$

(13)

where:

- $\text{TASKS}_\text{MX}[\hat{t}] = T_i$ for tasks classification, where $\hat{t}$ denotes a task class, $\hat{t} = 1, \ldots, T$, and $T_i$ denotes a fraction of tasks in the class $\hat{t}$. That is to say:
  $$T_i = \frac{\hat{t}}{n}$$

(14)

where $\hat{t}$ is the number of tasks in the class $\hat{t}$ and $n$ is the number of tasks in a given batch, and

$$\sum_{t=1}^{T} T_t = 1$$

(15)

- $\text{MACHINES}_\text{MX}[\hat{r}] = R_i$ for resources classification, where $\hat{r}$ specifies a machine class $\hat{r} = 1, \ldots, R$, and $R_i$ expresses a
proportion of machines in the class $\hat{r}$. That is to say:

$$R_{r} = \frac{r_{r}}{m}, \quad (16)$$

where $\sum_{r=1}^{R} R_{r}$ is the number of machines in the class $\hat{r}$ and $m$ is the number of resources available for a given batch, and

$$\sum_{r=1}^{R} R_{r} = 1. \quad (17)$$

Based on the results of monitoring the machine failures and successful execution of tasks on machines generated by the grid simulator we can classify the output. For this purpose each machine class $k$ we select the unique major class of tasks $\tau(k)$, which contains the greatest number of successfully executed tasks. The output data for the network can be represented by a matrix $OUT\_MX$ of the size $T \cdot R$ and $R$ positive elements (to indicate one major class of tasks per each host), where $OUT\_MX[k][\tau(k)] = r_{\hat{r}(k)}$, and $r_{\hat{r}(k)}$ is the proportion of the tasks from a major class $\tau(k)$ submitted to the machines of the class $\hat{r}$. The main concept of the network is presented in Fig. 4.

The network is trained by the back propagation algorithm [14] 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 in the MCT algorithm for generating the most promising solutions.

We select a task from a batch and check for it the ANN ‘suggestion’ by analyzing the network output. If the task is from the major class $\tau(k)$, then we choose the class of machines which are the best for a given tasks in the sense of the ANN output. Thereafter, we assign this task to the machine from a selected class with a minimal completion time using MCT procedure. If the selected task does not belong to the major class, then it is assigned to the machine with the shortest completion time according to the MCT procedure without analyzing the network output. The general framework of MCT is presented in Algorithm 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

Both input and output of ANN are totally independent of number of hosts or tasks in the system, thus the proposed ANN can be trained even on a small batch of task and small cluster of machines and then the generated output can be used in more complex scenarios.

The definition of just one major class of tasks for the network output is of course the drawback and the limitation of the proposed ANN approach. We would like to extend it to the multi-class task output generation in our future research.

5. Security aware genetic-based batch schedulers

Scheduling in CGs still remains a challenging, very complex and computationally hard global optimization task, mainly because of the multi-constraints and different usually also conflicting optimization criteria in the dynamic highly distributed environments. Heuristic methods are well known from their robustness and have been applied successfully to solve scheduling problems and general combinatorial optimization problems in a variety of fields [15–19]. Therefore they can be considered as good candidates to be the effective CG schedulers that tackle the various scheduling attributes and additional security aspects.

The heuristic grid schedulers are usually classified into three main groups, namely: (i) calculus-based (greedy algorithms and ad-hoc methods), (ii) stochastic (guided and non-guided methods), and (iii) enumerative methods (dynamic programming and branch-and-bound algorithm). One of the most popular and efficient class is created by the population-based methods.

In this work, we consider six genetic-based risk resilient grid schedulers presented in Table 1 that work in both risky and secure scheduling scenarios explained in Section 3.3.2.

The above 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. The general $= \pm$ steps procedure of calculating the fitness function for all algorithms is defined in Section 3.3.1. However, the completion times used there for calculating the makespan and mean flowtime depend on the mode in which the scheduler is working and are specified by formulas (11) and (12) for risky and secure scheduling, respectively. The ANN module is active just in GA-SS-ANN and GA-ST-ANN algorithms for generating a part of an initial population. All of the remaining procedures in these algorithms are identical with the schedulers working in the secure scenario.

We define in Algorithm 2 the genetic engine of all considered schedulers, which is based on the framework of the classical genetic algorithms used in the combinatorial optimization [20]. This method was adapted to the CG scheduling problem through an implementation of specialized encoding methods and genetic operators.

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

1: Generate the initial population $P^{0}$ of size $\mu$; $t = 0$
2: Evaluate $P^{t}$
3: while not termination-condition do
4: Select the parental pool $T^{\lambda}$ of size $\lambda$; $T^{\lambda} := Select(P^{t})$;
5: Perform crossover procedure on part of individuals in $T^{\lambda}$ with probability $p_{c}$; $P_{c}^{t} := Cross(T^{\lambda})$;
6: Perform mutation procedure on individuals in $P_{c}^{t}$ with probability $p_{m}$; $P_{m}^{t} := Mutate(P_{c}^{t})$;
7: Evaluate $P_{c}^{t}$;
8: Create a new population $P^{t+1}$ of size $\mu$ from individuals in $P_{c}^{t}$ and $P_{m}^{t}$; $P_{c}^{t+1} := Replace(P_{c}^{t}, P_{m}^{t})$
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 $P_{c}^{t}$ and $P^{t+1}$, and permutation representation in $P_{m}^{t}$ and $P_{c}^{t+1}$ populations to implement the mutation and crossover operators.

In the algorithms working in risky and secure modes 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 [21]. In LJFR-SJFR method, initially, the
number of \( m \) tasks with the highest workload are assigned to the available \( m \) machines sorted under the computing capacity criterion. Thereafter, the remaining unassigned tasks are allocated in the fastest available machines. 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).

For configuration of genetic operations in the main loop of the Algorithm 2 we used the following genetic operators especially designed for the combinatorial optimization [20]:

- **Selection operators**: Linear Ranking;
- **Crossover operators**: Cycle Crossover (CX);
- **Mutation operators**: Move;
- **Replacement operators**: Steady State, Struggle.

In Cycle Crossover (CX) that each task in a chromosome must occupy the same position, so that only interchanges between alleles (positions) can be made. Firstly, a cycle of alleles is identified. The cycle operator leaves the cycles unchanged, while the remaining segments of the parental strings are exchanged. The main idea of Move mutation a task is moved from one machine to another one. Although the task can be appropriately chosen, this mutation strategy tends to unbalance the number of tasks per machine.

We consider two alternate replacement mechanism for the generation of the base population for a new GA loop, namely Steady State and Struggle strategies. In the steady state method, a set of the best offsprings 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. On one hand this property is usually interesting to explore for grid scheduling problems because of the significance of the fast reductions of the makespan. But on the other hand, the high probability of the premature convergence can be the reason of the low effectiveness of the scheduler in the exploration of the new regions in the optimization landscape and very fast unification of the populations.

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 definition of the struggle replacement procedure requires a specification of the appropriate similarity measure, which indicates the degree of the similarity among two GA’s chromosomes. We use in this work the Mahalanobis distance [22] for measuring the distances between schedules according to the following formula:

\[
sim_{\text{e}}(s_1; s_2) = \frac{1}{\sigma_P^2} \sum_{j=1}^{n} (s_1[j] - s_2[j])^2
\]

where \( \sigma_P \) is the standard deviation of the \( s_{1}[j] \) over the population \( P \).

The struggle strategy has shown to be very effective in solving several large-scale multi-objective problems (see e.g., [17,23]). However, the computational cost can be very high, because of the need of calculation of distances among all off springs 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 [12]. 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 \( f_{\text{hash}} \) is defined as follows:

\[
f_{\text{hash}}(K) = \begin{cases} 
0, & K < K_{\text{min}} \\
N \cdot \left( \frac{K - K_{\text{min}}}{K_{\text{max}} - K_{\text{min}}} \right), & K_{\text{min}} \leq K < K_{\text{max}} \\
N - 1, & K \geq K_{\text{max}}
\end{cases}
\]

where \( K_{\text{min}} \) and \( K_{\text{max}} \) correspond respectively to the smallest and the largest value of \( K \) in the population, and \( N \) is the population size. Using the struggle replacement mechanism in genetic Grid schedulers allow us a fine tuning of the scheduler to ‘converge’ to a good solution depending on available time (for instance, scheduler’s time activation interval).
6. Experimental evaluation

In this section we present a simple experimental analysis of the influences of the ANN monitoring support on the performance of the scheduling metaheuristics in the case of the individual security requirements of the grid end users. This analysis has been realized in three main stages, namely (i) the comparison of the makespan and flowtime values for ANN-supported and non-supported genetic schedulers, (ii) the analysis of the influence of the ANN support on the reduction of the run time of the schedulers, and (iii) the analysis of the influence of the ANN module in the reduction of the machine failure rates. We conducted our experiments for six genetic-based schedulers defined in Table 1 in static and dynamic grid environments by using the HyperSim-G simulator.

6.1. Security aware HyperSim-G grid simulator—basic concept

To simulate the secure scheduling we define a Secure HyperSim-G simulator by extending the HyperSim-G framework [11]. 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.

The main concept of the security aware HyperSim-G simulator is presented in Fig. 5.

As we can see in Fig. 5, the simulator generates an instance of the problem based on 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 sub-optimal schedule, which is copied to the initial population of a GA-based scheduler. The ANN module is not considered in this work in fact as another (separate) scheduling method. It works in a “background” of the main process.

The instances produced by the simulator for our experiments are divided into static and dynamic grid scheduling benchmarks. In the static case, the number of tasks and the number of machines remain constant during the simulation, while in the dynamic case, both parameters may vary over time.

6.2. Experimental settings and performance metrics

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

We considered the following four Grid size scenarios in our study: (a) small grid (32 hosts/512 tasks), (b) medium grid (64 hosts/1024 tasks), (c) large grid (128 hosts/2048 tasks), and (d) very large grid (256 hosts/4096 tasks).

In the dynamic case, we have to specify the minimal and maximal values for numbers of tasks and machines in the system. The resources can be dropped or added to grid with the frequencies defined by the Gaussian distributions (add host and delete host parameters). New tasks may arrive in the system with frequency parameter denoted by interarrival, until a total tasks value is reached.

In the ANN module, the tasks and machines are divided into 18 classes described in Section 4. 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.

\(^2\) We use the notation \(U[x, y]\), \(N(a, b)\) and \(E(c, d)\) for uniform, Gaussian and exponential probability distributions respectively.
The key parameters for all types of genetic-based schedulers are presented in Table 3.

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

-**Makespan** (see Eq. (6)) for secure and risky scenarios,
-**Flowtime** (see Eq. (7)) for secure and risky scenarios, and
-**Failure Rate $F_f$** parameter defined as follows:

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

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.

### 6.3. Results

#### 6.3.1. Makespan and Flowtime Optimization

In the first part of the experimental analysis we just monitored and measured the best achieved makespan and flowtime values for all considered schedulers. We wanted to verify our hypothesis, that ANN through its ‘suggestions’ of the task–machine mappings made based on schedule execution monitoring, may improve the effectiveness of the genetic schedulers in the sense of the minimization of both major scheduling criteria.

In Figs. 6 and 7 we present the results of our experiments for six genetic-based risk resilient schedulers defined in Section 5 (see Table 1).

The best makespan results have been achieved by the ANN-supported schedulers, especially in the case of ‘large’ and ‘very large’ grid architectures. The worst in the makespan reduction were the schedulers working in the risky mode, the results strongly differ from the others (more than 20% of the difference). It is also important that for different test scenarios, the distribution of the results are asymmetric and the medians are very close to the first or the third quantiles. In the risky mode, the spread of values is significant and reaches about 30% between the minimum and maximum.

In the case of flowtime optimization, the differences for the various scenarios (between the first and the third quantiles) are rather small (not exceeding 10%), and the distribution is symmetrical. Similarly as in the makespan case, the best results have been achieved by the GA-XX-ANN schedulers, and the worst—by the GA-XX-R. Together with the raise of the size of the grid systems (the size is expressed in numbers of the active machines in the grid network), the flowtime values increase considerably for all applied schedulers, while the makespan values are kept almost at the same level. The best relative effectiveness of the ANN support may be observed in ‘very large’ grids in static and dynamic cases.

#### 6.3.2. Schedulers’ Run Time

Another aim of our experimental analysis is the verification of the influences of ANN support in the optimization of run times of the schedulers. Such ‘run times’ for genetic algorithms are usually measured in the number of the genetic epochs defined as a stopping criterion or necessary for detection of the global solution (i.e. the best schedule). In this case we also expected that ANN ‘suggestions’ of the potential sub-optimal schedules may reduce the run time of the supported schedulers. Our research also contains the impact of ANN “suggestions” on the number of epochs needed to obtain the best schedule. Table 4 presents the average numbers of genetic epochs\(^3\) necessary for the generation of the best achieved schedulers. The stopping criteria for all schedulers have been defined by a priori as $5 \cdot n$, where $n$ denotes the number of tasks in the system (see Table 3). In parentheses we expressed the values of 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.

---

3 We averaged the results generated in 30 runs of the simulator under the same configuration of the parameters.
It can be observed from the results that the activation of ANN module in most of the grid size scenarios allows to reduce the number of epochs and therefore also the run time of the schedulers needed for generating the best solution, by 30%–40%, which is significant improvement of the schedulers’ performances.

6.3.3. Reliability of the resources

In the last part of our experimental analysis we focused on monitoring and comparison of the failures of the resources in during the scheduling process due to insufficient security conditions and data and task protection. In Table 5 we compared the values of the failure rates of the machines calculated for all schedulers in all grid scenarios. In most of the cases, the schedulers supported by ANN module achieved the lowest failure rates. Indeed, the sub-optimal solutions generated by ANN module and integrated with the schedules’ populations allow to reduce the machine failures by 1%–6%. It can be also observed that it is more resilient for the grid scheduler to pay some additional scheduling ‘cost’ due to verification of the security conditions instead of taking a risk on unreliable resources allocated. As we expected, the failure rates in the risky mode are much higher than in the secure case, especially in the dynamic grid where the frequency of the machine failures are 3–4 times greater than in the secure scenario.

7. Related work

Monitoring of the grid and cloud resource utilization is a hot research topic over the last decade. An integrated monitoring infrastructure for Cloud environments is presented in [24]. The authors focus on the distribution of the infrastructure, the scalability of the applications and the energy consumption metrics as a main monitoring areas. Other approach is presented in [25], where the resource management was indicated as a key issue in grid computing. To support the constantly changing states of grid resources a Grid Resource Information Monitoring prototype and Grid Resource Information Retrieving protocol were proposed. The authors achieved a high degree of data accuracy and significant reduction of the cost of updating states.

The results of the monitoring of the scheduling process depends on the type and effectiveness of the schedulers, which is difficult in the case of the special user requirements on the system and resource secure access and task and data processing. Significant
Fig. 7. The box-plot of the results for flowtime in (a) static and (b) dynamic cases.

Table 4
The number of genetic epochs necessary for the generation of the best solutions (efficiency parameter $\eta$) found by six considered GA-based schedulers.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
<th>Very large</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static instances</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA-SS-R</td>
<td>2302</td>
<td>4722</td>
<td>10008</td>
<td>19226</td>
</tr>
<tr>
<td>GA-SS-S</td>
<td>2031</td>
<td>3620</td>
<td>8345</td>
<td>19740</td>
</tr>
<tr>
<td>GA-SS-ANN</td>
<td>1722</td>
<td>2733</td>
<td>7992</td>
<td>17739</td>
</tr>
<tr>
<td>GA-ST-R</td>
<td>1923</td>
<td>4213</td>
<td>10013</td>
<td>20054</td>
</tr>
<tr>
<td>GA-ST-S</td>
<td>1987</td>
<td>4005</td>
<td>8022</td>
<td>18654</td>
</tr>
<tr>
<td>GA-ST-ANN</td>
<td>1592</td>
<td>3872</td>
<td>8591</td>
<td>17910</td>
</tr>
<tr>
<td><strong>Dynamic instances</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA-SS-R</td>
<td>2090</td>
<td>5099</td>
<td>10100</td>
<td>20145</td>
</tr>
<tr>
<td>GA-SS-S</td>
<td>1831</td>
<td>3925</td>
<td>9036</td>
<td>19002</td>
</tr>
<tr>
<td>GA-SS-ANN</td>
<td>1622</td>
<td>3021</td>
<td>8010</td>
<td>17830</td>
</tr>
<tr>
<td>GA-ST-R</td>
<td>1987</td>
<td>4005</td>
<td>8022</td>
<td>18654</td>
</tr>
<tr>
<td>GA-ST-S</td>
<td>1703</td>
<td>2954</td>
<td>8238</td>
<td>17993</td>
</tr>
<tr>
<td>GA-ST-ANN</td>
<td>1511</td>
<td>3401</td>
<td>6035</td>
<td>17910</td>
</tr>
</tbody>
</table>

volume of research has been done in the domain of security and resource reliability in cloud computing and the cloud physical layer such as CGs [26,27]. The security-aware scheduling process in grid environment [28–30] becomes more challenging than scheduling problems defined for supercomputers, real-time, and parallel computers [31–33]. Unfortunately, well-known scheduling approaches for grid computing largely ignore this security factor, with only a handful of exceptions.
Humphrey and Thompson presented [28] a classification of security-aware grid models for an immediate job execution mode. They define a job control system for accessing Grid information services through authentication. However, they did not elaborate on how a scheduler should be designed to address the security concerns in collaborative computing over distributed cluster environment. An extensive survey of the research endeavors in this domain is presented in [34]. In [35] the authors present an approach on fault-tolerance method in CG scheduling. They provided a failure detection service, which enables the detection of both task failures and user secure requirements, and a flexible failure handling framework as a fault-tolerant mechanism on the grid. Abawajy [36] developed a model, in which jobs are replicated at multiple grid sites to improve the probability of the satisfaction of the security requirements and successful job executions.

The most promising recent security-aware approaches in CG scheduling are based on the game-theoretical models. In [6,5] the authors considered the risky and insecure conditions in online scheduling in CGs caused by software vulnerability and distrusted security policy. They apply the game model introduced in [37] for simulating the resource owners selfish behavior. The results presented in [5] are extended by Wu et al. in [38]. The authors consider the heterogeneity of fault-tolerance mechanisms in a security-assured Grid job scheduling and define four types of GA-based online schedulers for the simulation of fault-tolerance mechanisms. The other game-theoretical approaches are presented in [7]. In all of the aforementioned models the final decisions on the secure allocation of task to resources are made by the CG users who do not cooperate with each other.

Also it is worth to note Pop et al. paper [1]. The authors present a decentralized genetic-based scheduling algorithm and Improved Critical Path with Descendent Prediction (ICPDP) algorithm for DAG scheduling in Grid environments. ICPDP algorithm has been used to describe the fitness function. The DAG Scheduling algorithm achieved the best schedule length over all resources in a shorter time than the others. Moreover, the distribution of the loads is characterized by a good load balancing and efficient resource allocation. It was possible by minimizing the idle times on the processing elements.

Artificial Intelligence methods are employed more often in grid computing as a support mechanism. In [39] the authors indicate the need for flexible mechanisms that release from the decision-making by user and allow making intelligence auto-decisions. There are some works that take this subject. In [40] the authors present the scheduling algorithm based on Fuzzy Neural Networks. In their approach the fuzzy logic module is responsible for the evaluation of the grid system load status. Then the ANN trained through the back-propagation algorithm tunes the parameters of the fuzzy membership functions. An interesting ANN support to grid resource allocation is demonstrated in [41]. The authors use the Extreme Learning Machine (ELM) mechanism in a single-hidden layer feed-forward neural network (SLFN) with arbitrarily selected input weights and hidden neuron biases. The output weights of ELM are analytically calculated by Moore–Penrose generalized inverse.

The application of the ANN as the decision making mechanism for the grid users is presented in [2]. The authors introduced the users decision-based scheduling, the artificial neural network paradigm. Their model consists of three main components: (a) online module dedicated to the prediction of the users actions; (b) off-line model based on the analysis of statistical data acquired during users work; and (c) change detection module defined for the detection of trends and changes in users’ activities. The users’ decisions mechanism is based on the feed-forward neural networks trained by the back-propagation method. This method can be considered as an alternative to game-based ones for the characteristics of the dynamics of the Grid users’ decision process in the online scheduling. The authors additionally proposed the off-line model, where another neural network is applied, for the detection of normal/abnormal users activities, by analyzing the statistical data accumulated during the users actions. It seems that the proposed approach can be a good start for some more detailed analysis of the users decision processes, however the complexity of the model can be a main drawback for its successful application in real-life Grid scenarios.

Artificial Neural Networks have been not so widely applied for the monitoring of the cloud systems and for the prediction of the resource utilization and improvement of the task–machine mapping. Duy et al. in [42] described a Green Scheduling Algorithm integrated with a neural network predictor for optimizing server power consumption in Cloud computing. According to predicted future load demand, the algorithm turns off unused servers and restarts them to minimize the number of running servers. As a result, the energy use up to 46% was reduced.

To conclude, it seems that application of AI methods in monitoring and scheduling support pin high-performance distributed systems with additional security conditions is a quite innovative approach.

### 8. Conclusions

In this paper, we presented the results of our recent research on monitoring the scheduling efficiency and resource reliability in highly distributed computing environments. We developed a security-aware scheduling model with the Artificial Neural Network (ANN) support for monitoring the system reliability and decision-making activities of grid actors (end users, schedulers, system administrators and resource providers). We defined the trust levels parameters for resources and security demands for tasks in order to estimate the probability of resource failures during the task execution. ANN module is an external component of the system and monitors the scheduling and task execution processes. The neural network learns from the observed failures of the machines and generates the task–machines mapping suggestions. Those “suggestions” are considered then as sub-optimal schedules and are sent to the schedulers as possible alternate solutions of the scheduling problem. In our case, such schedules have been moved to the populations of schedules generated for genetic-based scheduling metaheuristics.

We have developed six risk-resilient genetic-based schedulers and integrated them with HyperSim-G Grid Simulator. We used the Minimal Completion Time (MCT) algorithm for the sub-optimal schedule generation by the ANN module. Compared to our preliminary results presented at 2014 European Conference on Modelling and Simulation in Brescia in May 2014, we considered in the experimental study four (instead of two) grid infrastructure scenarios, namely grids with 32, 64, 128 and 256 distributed hosts. We also

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
<th>Very large</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static instances</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA-SS-R</td>
<td>4.832</td>
<td>7.201</td>
<td>11.824</td>
<td>31.721</td>
</tr>
<tr>
<td>GA-SS-5</td>
<td>4.008</td>
<td>4.135</td>
<td>10.698</td>
<td>11.635</td>
</tr>
<tr>
<td>GA-SS-ANN</td>
<td>3.993</td>
<td>4.089</td>
<td>5.436</td>
<td>6.736</td>
</tr>
<tr>
<td>GA-ST-R</td>
<td>4.697</td>
<td>17.516</td>
<td>14.013</td>
<td>35.643</td>
</tr>
<tr>
<td>GA-ST-S</td>
<td>3.897</td>
<td>5.540</td>
<td>10.945</td>
<td>7.402</td>
</tr>
<tr>
<td>GA-ST-ANN</td>
<td>3.967</td>
<td>6.43</td>
<td>4.11</td>
<td>8.196</td>
</tr>
<tr>
<td><strong>Dynamic instances</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA-SS-5</td>
<td>6.104</td>
<td>6.916</td>
<td>5.507</td>
<td>8.943</td>
</tr>
<tr>
<td>GA-SS-ANN</td>
<td>4.884</td>
<td>5.667</td>
<td>4.756</td>
<td>5.026</td>
</tr>
<tr>
<td>GA-ST-R</td>
<td>26.797</td>
<td>22.96</td>
<td>29.227</td>
<td>29.380</td>
</tr>
<tr>
<td>GA-ST-S</td>
<td>9.218</td>
<td>7.623</td>
<td>5.084</td>
<td>5.744</td>
</tr>
<tr>
<td>GA-ST-ANN</td>
<td>4.093</td>
<td>5.097</td>
<td>4.681</td>
<td>7.894</td>
</tr>
</tbody>
</table>
References


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. For more information, please visit: http://www.grzonka.eu/.

Joanna Kołodziej graduated in Mathematics from the Jagiellonian University in Cracow in 1992, where she also received the Ph.D. in Computer Science in 2004. She currently works as an associate professor at Cracow University of Technology and is a Vice Head of the Computer Science Department responsible for research and development. She has served and is currently serving as PC Co-Chair, General Co-Chair and IPC member of several international conferences and workshops including IPDPS, ECMS, HPCC and many others. She 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. For more information, please visit: http://www.joannakolodziej.org/.

Jie Tao got her Ph.D. degree from the department of Computer Science at Munich University of Technology, Germany. She is currently a senior research associate at the Steinbuch Centre for Computing, Karlsruhe Institute of Technology. Her research work targets mainly on parallel and distributed computing, data-intensive computing as well as Grid and Cloud computing. She 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.

Samee Ullah Khan received a BS degree in 1999 from Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, and a Ph.D. in 2007 from the University of Texas, Arlington, TX, USA. Currently, he is Associate Professor of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA. His research interests include optimization, robustness, and security of: cloud, grid, cluster and big data computing, social networks, wired and wireless networks, power systems, smart grids, and optical networks. His work has appeared in over 250 publications. He is on the editorial boards of leading journals, such as IEEE Transactions on Computers, IEEE Access, IEEE Cloud Computing Magazine, IEEE Communications Surveys and Tutorials, Scalable Computing, Cluster Computing, Security and Communication Networks, and International Journal of Communication Systems. He is a Fellow of the Institution of Engineering and Technology (IET, formerly IEE), and a Fellow of the British Computer Society (BCS). He is a Senior Member of the IEEE. For more information, please visit: http://sameekhan.org/.