Chapter 4
Energy and Security Awareness in Evolutionary-driven Grid Scheduling

Joanna Kołodziej, Samee U. Khan, Lizhe Wang, Dan Chen and Albert Y. Zomaya

Abstract Ensuring the energy efficiency, thermal safety, and security-awareness in today's large-scale distributed computing systems is one of the key research issues that leads to the improvement of the system scalability and requires researchers to harness an understanding of the interactions between the system external users and the internal service and resource providers. Modeling these interactions can be computationally challenging especially in the infrastructures with different local access and management policies such as computational grids and clouds. In this chapter, we approach the independent batch scheduling in Computational Grid (CG) as a three-objective minimization problem with *Makespan*, *Flowtime* and energy consumption in risky and security scenarios. Each physical resource in the system is equipped with Dynamic Voltage Scaling (DVS) module for optimizing the cumulative power energy utilized by the system. The effectiveness of six genetic-based single- and multi-population grid schedulers has been justified in comprehensive empirical analysis.

Joanna Kołodziej
Institute of Computer Science, Cracow University of Technology, ul. Warszawska 24, 31-155 Cracow,Poland
e-mail: jkolodziej@uck.pk.edu.pl

Samee U. Khan
Department of Electrical and Computer Engineering, North Dakota State University, ND 58108, USA;
e-mail: samee.khan@ndsu.edu

Lizhe Wang
Center for Earth Observation and Digital Earth; Chinese Academy of Sciences; Beijing; China
e-mail: LZWang@ceode.ac.cn

Dan Chen
China University of Geosciences Wuhan, China;
e-mail: Danjj43@gmail.com

Albert Y. Zomaya
School of Information Technologies, University of Sydney, Sydney, NSW 2006, Australia;
e-mail: albert.zomaya@sydney.edu.au

4.1 Introduction

The concept of today's grid systems grown far beyond the original models of high performance computing centers and networks. The modern Computational Grids (CGs) are designed as the complex infrastructures that are made up of hundreds or thousands of various components (computers, databases, etc) and numerous user-oriented services and procedures. The management of such infrastructures remains challenging problem, especially when additional personalization of the communication protocols and security of the transferred data and information are the crucial issues. We can observe significant disproportion of resource availability and resource provisioning in such systems and the increase in energy consumption to match resource provisioning in light of the computational demands [25]. Therefore, the ability of the efficient scheduling of the grid applications and the allocation of the resources in a desired configuration of all system components in a scalable and robust manner is essential in today's grid computing.

The intelligent grid schedulers should efficiently optimize the standard scheduling objectives, such as *Makespan*, *Flowtime* and resource utilization, but also can meet the security requirements of the grid users and can minimize the energy consumed by all of the system components. Energy-efficient and security-aware scheduling in CGs becomes complex endeavors due to the multi-constraints and different optimization criteria and different priorities of the resource owners. The computational intractability of the problems calls for heuristic approaches as effective means for designing risk-resilient energy-aware grid schedulers by trading-off among the different scheduling requirements, constraints and scenarios. Recent literature leverages the capability of Genetic Algorithms (GAs) to provide satisfactory green computing solutions as well as security-aware scheduling by tackling the aforementioned scheduling challenges.

This chapter addresses the *Independent Batch Job Scheduling* problem in CGs, where tasks are processed in a batch mode, there are no dependencies among tasks, and two standard scheduling objective functions, namely *Makespan* and *Flowtime*, are minimized along with security and cumulative energy consumption in the system. A comprehensive empirical analysis of wide range of single- and multi-population genetic-based metaheuristics has been provided in static and dynamic grid environments by using the grid simulator.

The following notation for tasks and machines in independent grid scheduling is introduced from this point forward will be used throughout the chapter:

- n – the number of tasks in a batch;
- m – the number of machines available in the system for the execution of a given batch of tasks;
- $N = \{1,\ldots,n\}$ – the set of tasks' labels;
- $M = \{1,\ldots,m\}$ – the set of machines' labels.

The rest of the chapter is organized as follows. The generic architecture of the security-aware grid is defined in Section 4.2. The main scheduling attributes along

with the definitions of scheduling problems, scheduling scenarios and main objectives are specified in Section 4.4. The generic framework of the single-population grid schedulers and various combinations of the genetic operators are presented in Section 4.5. The empirical evaluation of 18 types of simple genetic schedulers is provided in Section 4.6. Sec. 4.7 describes the main concepts and the empirical analysis of the effectiveness of four types of multi-population genetic strategies in grid scheduling. Related work is discussed in Section 4.8. The chapter is concluded in Section 4.9.

4.2 Generic Model of Secure Grid Cluster

Grid system is usually modelled as a multi-layer architecture with the hierarchical management system that consist of two or three levels depending on the system knowledge, access to data and the system services and resources. The whole architecture is composed of the local computational clusters, as it is presented in Fig. 4.1, and it is defined as a compromise between the centralized and decentralized resources and service managements.

The exemplary two-level architecture is a simple Meta-Broker model presented in [14]. In this model the Meta-Broker (MB) operates in the inter-site global level and is responsible for the processing of all applications and information submitted by the grid users to the resource administrators. He also collects the feedback information from the resource owners and send the results of the computations to the users.

A three-level grid system example is presented by Kwok et al. in [33]. I this model there is a central global authority (central scheduler) who is responsible for the final optimal assignment of the tasks submitted to the system and the communication with the external grid users. The local system managers communicate with the resource administrators and resource owners and specify the "grid sites reputation indexes" that are further forwarded to the global scheduler. All the machines and resource providers work at the local intra-site level.

The hierarchical structure of the system management must be additionally combined with the multi-layer structure of the main grid components [10], namely: (1) grid "fabric" layer, (2) grid core middleware, (3) grid user layer, and (4) applications layer. The grid "fabric" layer is composed of the grid resources, services, and local resource management systems. The grid core middleware provides services related to security and access management, remote job submission, storage, and resource information and scheduling. The grid user layer contains all of the grid service end-users and system entities, and plays the most important role in ensuring multi-criterion scheduling and resource management efficiency.

The roles of the meta-scheduler and meta-brokers in grid clusters are different when security is considered as additional criterion in the scheduling process. The meta-scheduler must analyze the security requirements defined as security demand parameters for the execution of tasks and requests of the CG users for trustful re-

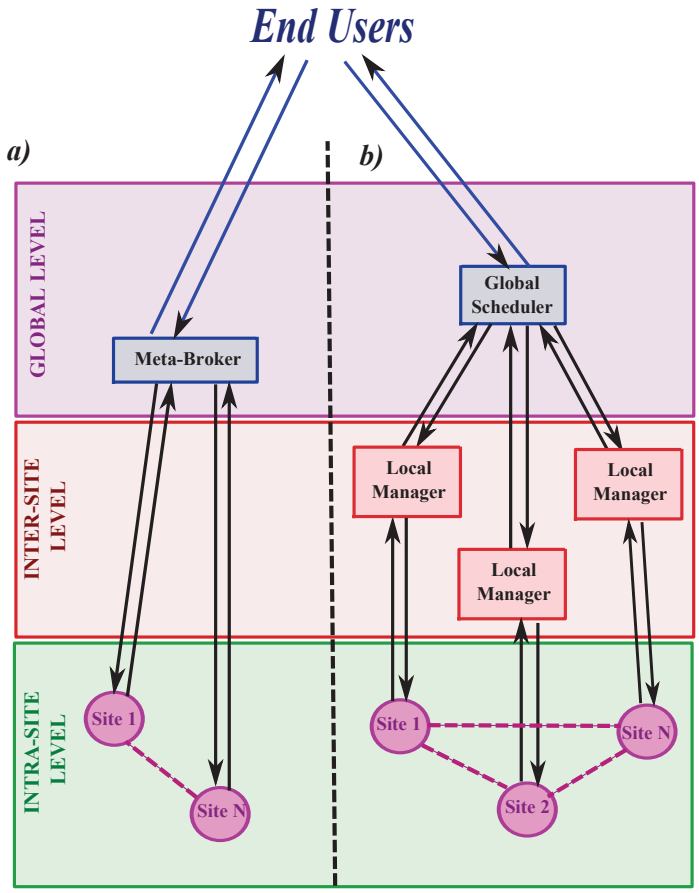

Fig. 4.1: Two-level (a) and three-level (b) hierarchical architecture of clusters in Computational Grids

sources available within the system. The system brokers analyze "trust level" indexes of the machines received from the resource managers and send proposals to the scheduler. Moreover, the brokers also control the resource allocation and communication between CG users and resource owners.

The trust level and security demand parameters are the results of the aggregation of several scheduling and system attributes [47]. The main attributes can be specified as follows:

- **Security Demand (Tasks) Attributes:**
  - data integration;
  - task sensitivity;
  - access control;

  - peer authentication;
  - task execution environment;
- **Trust Level (Machines) Attributes:**
  - *Behavioral Attributes:*
    - prior task execution success rate;
    - cumulative grid cluster utilization;
  - *Intrinsic Security Attributes:*
    - firewall capabilities;
    - intrusion detection capabilities;
    - intrusion response capabilities.

The aggregation of the above mentioned attributes into a single-valued scalar parameters can be realized by using the fuzzy logic-based methods as it is demonstrated by Song et al. in [46]. In this model the machines and users' applications are characterized by the *security demand vector* $SD = [sd_1, \ldots, sd_n]$ and *trust level vector* $TL = [tl_1, \ldots, tl_m]$. 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. The model of Song et al. is applied for the characteristic of machines and tasks in grid scheduling problem defined in the following section (Sec. 4.3). The process of matching sd_j with tl_i is similar to that of a real-life scenario where users of some portals, such as Yahoo!, are required to specify the security level of the login session.

4.3 Scheduling problems in Computational Grids

There are numerous types of scheduling problems that can be specified in highly parametrized computational environments such as computational grids. A particular type of the problem can be defined with respect to different properties of the underlying grid environment and various requirements of the users, namely **(a)** type of the environment, **(b)** type of the grid architecture, **(c)** task processing policy, and **(d)** tasks' interrelations. To achieve the desired performance of the system all this information must be "embedded" into the scheduling mechanism [2], [29]. The aforementioned four scheduling attributes are presented in Fig. 4.2.

Depending on the type of the grid environment, the scheduling may be realized as the *static* or *dynamic* process. In the case of the static scheduling the number of the submitted applications and the available resources remain constant in a considered time interval, while in the dynamic scenario the resources may be added or removed from the system in an unpredictable way.

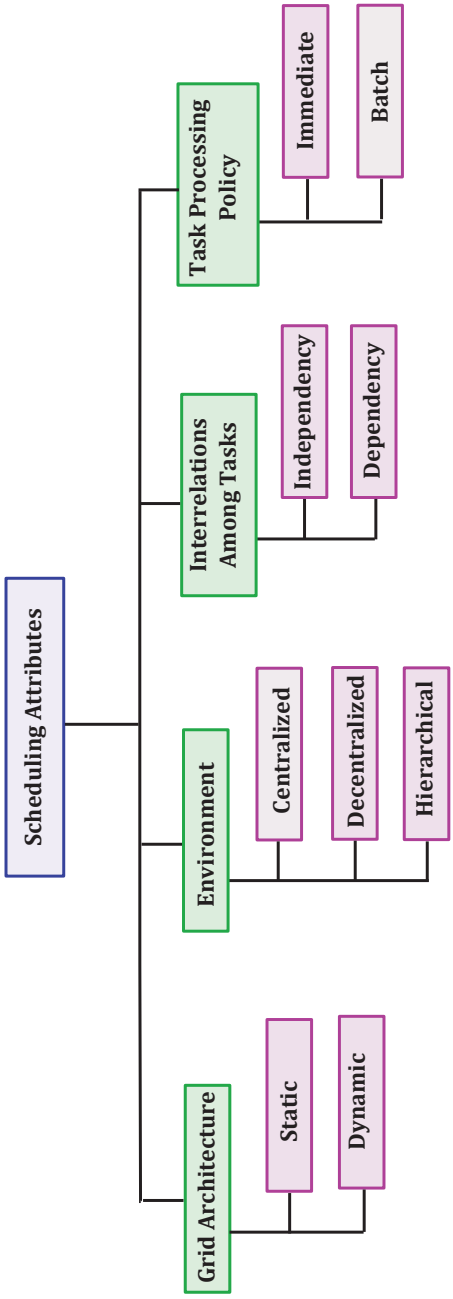

Fig. 4.2: Main scheduling attributes in Computational Grids

The resource management and scheduling can be organized in centralized, decentralized, or hierarchical modes. In *centralized model*, there is a central scheduler with a full knowledge of the system. In *decentralized model*, local schedulers interact with each other to manage the tasks submitted to the system. Finally, in the *hierarchical model*, there is a central meta-scheduler, which interacts with local managers (brokers) in order to define the optimal schedules.

The tasks in the system can be processed according to the *immediate* policy, where the tasks are scheduled as soon as they are entered into the system, or *batch* mode, where submitted tasks are grouped into batches and the scheduler assigns each batch to the resources.

Finally, tasks may be *independently* submitted and calculated in the grid system or considered as parallel applications with priority constraints and *interrelations* among the application components (usually modeled by a Directed Acyclic Graph (DAG)).

4.3.1 Problems Notation and Classification

To our best knowledge there is no standard unified notation for classification of the scheduling problems in grids. Fibich et al. proposed in [13] a simple extension of the Graham's [17] and Brucker's [9] classifications of conventional scheduling problems in order to adapt this methodology to the dynamic grid environment. Based on the idea presented in [13] and [27] and the main scheduling attributes specified in the previous section, the basic notation for the grid scheduling problem instances in CGs can be defined as follows:

$$\alpha|\beta|\gamma \tag{4.1}$$

where α characterizes the resource layer and grid architecture type, β specifies the processing characteristics and the constraints, and γ denotes the scheduling criteria.

(α) Resource Characteristics and Grid Architecture Type

According to the standard resource notation [17] the grid computational resources can be classify as *Rm* machines[1] with possible different speeds for different jobs. grid architecture type can be denoted by *C* for centralized, *D* for decentralized and $H(i)$ for hierarchical system, where i denotes the system levels. The following notation

$$Pm, H(2) \tag{4.2}$$

[1] For single CPU machine α=1; identical machines in parallel infrastructure – *Pm*, machines in parallel with different speeds – *Qm*, unrelated machines in parallel –*Rm*, and (flow—open—job) shop resources (Fm, Om, Jm)

is used for the representation of the 2-level hierarchical grid architecture with parallel identical machines in the system.

(β) Tasks Processing Mode, Tasks Interrelations, Static and Dynamic Scheduling Modes and Scheduling Constraints

The following notation can be used for setting the grid tasks' attributes and scheduling modes (see also Fig. 4.2):

- b – batch mode;
- im – immediate mode;
- dep – dependency among tasks ;
- $indep$ – independent scheduling;
- sta – static scheduling;
- dyn – dynamic scheduling;

The scheduling may be conducted under various constraints specified by all the grid users. The typical constraints include budget and deadline (d_j) limits for a given task j. An instant of the independent batch scheduling in dynamic mode with a limited deadline can be denoted as follows:

$$b, indep, dyn, d_j. \tag{4.3}$$

(γ) Scheduling Criteria and Objectives

The problem of scheduling tasks in CG is multi-objective in its general setting as the quality of the solutions can be measured using several criteria.

Two basic models are utilized in multi-objective optimization: hierarchical and simultaneous modes. In the *simultaneous mode* (*sim*) all objectives are optimized simultaneously while in the *hierarchical* (*hier*) case, the objectives are sorted *a priori* according to their importance in the model. The process starts by optimizing the most important objective and when further improvements are impossible, the second objective is optimized under the restriction of keeping unchanged (or improving) the optimal values of the first, and so on. It is very hard in grid scheduling to define or efficiently approximate the Pareto front, especially in dynamic scheduling, where it may extend very fast together with the scale of the grid and the number of the submitted tasks.

The basic scheduling criteria can be defined as global optimization criteria and include: *Makespan*, *Flowtime*, resource utilization, load balancing, matching proximity, turnaround time, total weighted completion time, lateness, weighted number of tardy jobs, weighted response time, etc [50].

In this chapter we consider two standard optimization criteria for grid scheduling, namely *Makespan* and *Flowtime*, that can be defined as follows:

- *Makespan* – is defined as the finishing time of the latest task and can be calculated by the following formulae:

$$C_{max} = \min_{S \in Schedules} \left\{ \max_{j \in Tasks} C_j \right\}, \tag{4.4}$$

  where C_j denotes the time when task j is finalized, $Tasks$ denotes the set of all tasks submitted to the grid system and $Schedules$ is the set of all possible schedules generated for the considered batch of the tasks (or current number of tasks submitted to the system at a given moment); and
- *Flowtime*, that is expressed as the sum of finalization times of all the tasks. It can be defined in the following way:

$$C_{sum} = \min_{S \in Schedules} \left\{ \sum_{j \in Tasks} C_j \right\}. \tag{4.5}$$

  The notation

$$hier, (C_{max}, C_{sum}) \tag{4.6}$$

means that *Makespan* and *Flowtime* are optimized in the hierarchical mode.

Makespan and *Flowtime* define the deadline and *Quality of Service (QoS)* criteria that are usually considered in most of the grid scheduling problems. To express the security and energy awareness, some additional scheduling objective functions must be specified. In Section 4.4 we define the *Independent Batch Scheduling* problem and *Expected Time to Compute* matrix model. Three main scheduling scenarios specified by setting the security and energy scheduling attributes are discussed. The main scheduling objectives are expressed in terms of completions times of machines.

4.4 Independent Batch Scheduling Problem, Scheduling Scenarios and Objective Functions

Independent Batch Scheduling is one of the simplest and fundamental scheduling models in computational grids. It is assumed in this model that: (a) tasks are grouped into batches; (b) tasks can be executed independently in a hierarchically structured static or dynamic grid environments. According to the notation introduced in Sec. 4.3.1 (see formula (4.1) an instance of the independent batch grid scheduling problem can be expressed as follows:

$$RmH(3)\left[\{b, indep, (stat, dyn), hier\}\right](objectives)), \tag{4.7}$$

where:

- Rm – Graham's notation references that tasks are mapped into (parallel) resources of various speed[2]
- $H(3)$ – denotes 3-level hierarchical grid management system;
- b – designates that the task processing mode is "batch mode"
- $indep$ – denotes "independency" as the task interrelation
- (sta, dyn) – indicates that both static and dynamics grid scheduling modes are considered
- $hier$ – references that the scheduling objectives are optimized in hierarchical mode
- $objectives$ – denotes the set of the considered scheduling objective functions.

In the case of energy-aware scheduling presented in this chapter there are three $objective$ functions, namely:

- C_{max} – denotes *Makespan* that is the dominant scheduling objective;
- C_{sum} – denotes *Flowtime* as the *Quality of Service (QoS)* objective;
- $E_I(E_{II})$ – denotes total energy consumption (E_I or E_{II} is selected depending on the scheduling scenario (see Sec. 4.4.3))

The schedules are realized in the ***secure*** and ***risky*** modes specified according to the security requirements defined for scheduling by the grid users or/and the system administrators and service and resource provides.

4.4.1 Expected Time To Compute (ETC) Matrix Model Adapted to Energy and Security Aware Scheduling in Grids

Independent batch scheduling in CG can be modelled by using the *Expected Time to Compute (ETC)* matrix model [4]. In the conventional version of this model tasks and machines are characterized by the following parameters:

(a) **Task** j:

– wl_j – workload parameter expressed in Millions of Instructions (MI)– $WL = [wl_1, \ldots, wl_n]$ is a *workload vector* for all tasks in the batch;

(b) **Machine** i:

– cc_i – computing capacity parameter expressed in *Millions of Instructions Per Second (MIPS)*, this parameter is a coordinate of a *computing capacity vector*, which is denoted by $CC = [cc_1, \ldots, cc_m]$;
– $ready_i$ – 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
$ready\_times = [ready_1, \ldots, ready_m]$.

[2] In independent grid scheduling it is usually assumed that each task may be assigned just to one machine.

Tasks can be considered as monolithic applications or meta-task with no dependencies among the components. The term "machine" is related to a single or multiprocessor computing unit or even to a local small-area network.

For each pair (j,i) of task-machine labels, the coordinates of WL and CC vectors are usually generated by using some probability distributions (we have used the Gaussian distribution [34] for the purpose of the empirical analysis presented later on in this chapter) and are used for an approximation of the completion time of the task j on machine i. This completion time is denoted by $ETC[j][i]$ and can be calculated in the following way:

$$ETC[j][i] = \frac{wl_j}{cc_i}. \tag{4.8}$$

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

This model is useful for a formal specification of *Makespan* and *Flowtime* criteria, that can be expressed in terms of completion times of machines. A *completion time completion*$[i]$ of the machine i is defined as the sum of the ready time parameters for this machine and a cumulative execution time of all tasks actually assigned to this machine, that is to say:

$$completion[i] = ready_i + \sum_{j\in Task(i)} ETC[j][i], \tag{4.9}$$

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

The *Makespan* C_{max} is expressed as the maximal completion time of all machines, that is:

$$C_{max} = \max_{i\in M} completion[i]. \tag{4.10}$$

The cumulative *Flowtime* for a given schedule is defined as a sum of workflows of the sequences of tasks on machines, that is to say:

$$C_{sum} = \sum_{i\in M} \left[ready_i + \sum_{j\in Sorted[i]} ETC[j][i] \right] \tag{4.11}$$

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

The completion times of machines and the times of successful executions of tasks on these machines can change if the security and energy optimization conditions are additionally considered. Therefore, the conventional ETC matrix model must be extended by incorporating the additional characteristics of tasks and machines in the system, and the modification of the methodologies of the generation of the ETC matrix.

4.4.2 Security Conditions

In security-aware scheduling all the system "actors", namely grid end-users, grid schedulers, cluster service and resource providers, system administrators and resource owners may specify their own requirements for the secure execution of tasks on the grid machines and secure access to the grid data and services. The most important users' requirements can be expressed in the following way [28]:

- **Access to Remote Data**: The input and output data specified by the user may be stored remotely. Therefore, users will need to provide the location of the remote data. If a ubiquitous wide-area file system is in operation on the grid, the user would only have to care about the location of files and data with respect to some root location under which they are stored.
- **Resource Specification**: The user may specify his individual requirements for the resources necessary in optimizing the execution times and costs of scheduling and computing tasks. The user may wish to target particular types of resources (e.g. SMP machines), but should not be concerned with the type of resource management on the grid, nor with the resource management systems on individual resources on the grid.
- **Resource reliability**: In some cases, the machines within the grid system could be unavailable due to high system dynamics or special policies of the resource owners. The user should be informed about the resource reliability in order to reduce the cost of possible resource failures or the abortion of executed tasks. In the case of resource failure the system administrators can activate re-scheduling or task migration procedures, and pre-emption policies.
- **Trustfulness of Resources**: The user may be required to allocate his tasks in the most trustful resources. Therefore the user should be able to verify the trust indexes of the resources and estimate the security demands for his tasks on the available resources.
- **Standardized authentication and authorization mechanisms requirements**: The users will likely utilize a standardized certificate authentication scheme. The certificates can be digitally signed by a certificate authority, and kept in the user's repository, which is recognized by the resources and resource owners. It is desirable for a certificate to be automatically created by the user's interface application during task submission.

The successful execution of tasks submitted to CGs may be impossible (or interrupted) if such requirements are very strong and the access to the resources is limited. On the other hand, the grid cluster or the grid resource may be not accessible to the global meta-scheduler or grid user when being infected with intrusions or by malicious attacks. It means that some, even very simple, authorization and authentication protocols and some anti-viruses protection mechanism are needed for efficient scheduling especially in the dynamic environment. In such cases the machines and task should be additionally characterized by the trust level tl_i and security demands sd_j parameters as it was specified in Sec. 4.2.

Let us denote by Pr_f a *Machine Failure Probability* matrix, that defines the probabilities of failures of the machines during the tasks executions $P_f[j][i]$ for each task-machine pair (j,i). These probabilities can be calculated by using the negative exponential distribution function, that is to say:

$$Pr_f[j][i] = \begin{cases} 0 & , sd_j \leq tl_i \\ 1 - e^{-\alpha(sd_j - tl_i)} & , sd_j > tl_i \end{cases} \tag{4.12}$$

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

The scheduler may initialize his work in two different modes: (a) *secure mode*, where he analyzes the elements of the *Machine Failure Probability* matrix in order to minimize the failure probabilities for task-machine pairs; and (b) *risky mode*, in which he performs an "ordinary" scheduling without any preliminary analysis of the security conditions, aborts the task scheduling in the case of machine failure, and reschedule this task at another resource.

Secure Mode

In this scenario all of the security and resource reliability conditions are verified for all task-machine pairs (j,i). The main goal of the meta-scheduler is to design an optimal schedule for which, beyond the other criteria, the probabilities of failures of the machines during the tasks executions will be minimal. It is assumed that additional "cost" of the verification of security assurance condition for a given task-machine pair may delay the predicted execution time of the task on the machine. This cost is approximately proportional to the probability of failure of the machine during the task execution. The completion time of the machine i in the secure mode is denoted by $completion^s[i]$ and can be calculated as follows:

$$completion^s[i] = ready_i + \sum_{\{j \in Tasks(i)\}} (1 + Pr_f[j][i]) ETC[j][i]). \tag{4.13}$$

The *Makespan* and *Flowtime* in this case can be expressed as follows:

$$C^s_{max} = \max_{i \in M} completion^s[i]. \tag{4.14}$$

$$C^s_{sum} = \sum_{i \in M} \left[ready_i + \sum_{j \in Sorted[i]} \left(1 + Pr_f[j][i]\right) \cdot ETC[j][i] \right]. \tag{4.15}$$

Risky Mode

In this scenario all secure and failing conditions are ignored. The scheduling process is realized as a two-step procedure. First, the scheduling is performed just by analyzing the **conventional** *ETC* matrix. If failures of machines are observed, then the unfinished tasks are temporarily moved into the backlog set of tasks. The tasks

from this set are re-scheduled according the rules specified for the secure mode. The total completion time of machine i ($i \in M$) in this case can be defined as follows:

$$completion^r[i] = completion[i] + completion^s_{res}[i], \tag{4.16}$$

where $completion[i]$ is calculated by using the Eq. (4.9), for tasks primarily assigned to the machine i , and $completion^s_{res}[i]$ is the completion time of machine i calculated by using the Eq. (4.13) for rescheduled tasks, i.e. the tasks re-assigned to the machine i from the other resources.

The Makespan and Flowtime in this mode can be calculated in the following way:

$$C^r_{max} = \max_{i \in M} completion^r[i]. \tag{4.17}$$

$$C^r_{sum} = \sum_{i \in M} \left[ready_i + \sum_{j \in Sorted[i]} ETC[j][i] + \sum_{j \in Sorted_{res}[i]} \left(1 + Pr_f[j][i]\right) \cdot ETC[j][i] \right]. \tag{4.18}$$

In the hierarchical optimization mode (see Eq. (4.7), parameter *hier*) where *Makespan* is defined as the dominant scheduling criterion, the *Flowtime* objective function should be minimized in both secure and risky scenarios subject to the following constraints:

- in the secure mode

$$ready_i + \sum_{j \in Sorted[i]} \left(1 + Pr_f[j][i]\right) \cdot ETC[j][i] \leq C^s_{max}) \; \forall i \in M; \tag{4.19}$$

- in the risky mode

$$ready_i + \sum_{j \in Sorted[i]} ETC[j][i] + \sum_{j \in Sorted_{res}[i]} \left(1 + Pr_f[j][i]\right) \cdot ETC[j][i] \leq C^r_{max} \; \forall i \in M. \tag{4.20}$$

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. $sd_j \leq tl_i$ for $i \in M$, $j \in N$), the completion times of machines in both *secure* and *risky* modes are identical with the completion times defined for standard independent scheduling problem, where it is assumed that each task *must* be successfully executed on each machine and no security requirements are analyzed.

4.4.3 Energy Model

The energy model presented in this chapter is based on the *Dynamic Voltage and Frequency Scaling (DVFS)* technique [25] that is used for adjusting the voltage supplies and frequencies of the grid computational nodes.

Each machine in the grid is equipped with a DVFS module for scaling its supply voltage and operating frequency. It has been assumed that the frequency of the

machine is proportional to its processing speed [37]. It follows from the Eq. (4.24) that the reduction of the supply voltage and frequency is directly correlated to the reduction of the energy utilization. Table 4.1 shows the typical parameters for 16 DVFS levels and three main "energetic" categories for machines.

Table 4.1: DVFS levels for three machine classes

	Class I		**Class II**		**Class III**	
Level	**Volt.**	**Rel.Freq.**	**Volt.**	**Rel.Freq.**	**Volt.**	**Rel.Freq.**
0	1.5	1.0	2.2	1.0	1.75	1.0
1	1.4	0.9	1.9	0.85	1.4	0.8
2	1.3	0.8	1.6	0.65	1.2	0.6
3	1.2	0.7	1.3	0.50	1.9	0.4
4	1.1	0.6	1.0	0.35		
5	1.0	0.5				
6	0.9	0.4				

The energetic class of machine i, ($i \in M$) is denoted by s^i and it is represented by the vector $Vr_{(i)}$ of DVFS levels, that can be specified as follows:

$$Vr_{(i)} = \left[(v_{s_0}(i), f_{s_0}(i)); \ldots; (v_{s_{l(max)}}(i), f_{s_{l(max)}}(i))\right]^T \tag{4.21}$$

where $v_{s_l}(i)$ refers to the voltage supply for machine i at level s_l, $f_{s_l}(i)$ is a scaling parameter for the frequency of the machine at the same level s_l, and l_{max} is the number of levels in the class s^i. The parameters $\{f_{s_0}(i), \ldots, f_{s_{l(max)}}(i)\}$ lie in the [0,1] range and should be interpreted as the relative frequencies of the machine i from class s^i at the $s_0, \ldots, s_{l(max)}$ DVFS levels.

The reduction of the machine frequency and its supply voltage can lead to the extension of the computational times of the tasks executed on that machine. For a given "task-machine" pair (j,i), the completion times for the task j on machine i at different DVFS levels in the class s^i can be interpreted as the coordinates of a vector $\widehat{ETC}[j][i]$ that is defined in the following way:

$$\widehat{ETC}[j][i] = \left[\frac{1}{f_{s_0}(i)} \cdot ETC[j][i], \ldots, \frac{1}{f_{s_{l(max)}}(i)} \cdot ETC[j][i]\right], \tag{4.22}$$

where $ETC[j][i]$ are the expected completion times for task j on machine i calculated by using the conventional ETC matrix model.

In energy-aware scheduling the completion times calculated for each pair (j,i) of task–machine labels in conventional ETC matrix (see Eq. (4.8)) should be replaced by the $\widehat{ETC}[j][i]$ vectors, that is to say:

$$\widehat{ETC} = \left[\widehat{ETC}[j][i][s_l]\right]_{n \times m \times s_{l(max)}}, \tag{4.23}$$

where $\widehat{ETC}[j][k][s_l]$ is the time necessary for the completion of the task j on machine i at the level s_l.

The DVFS model is based on the power consumption model employed in complementary metal-oxide semiconductor (CMOS) logic circuits [6]. In CMOS model, the capacitive power Pow_{ji} utilized by the machine i for computing the task j it is expressed as follows:

$$Pow_{ji} = A \cdot C \cdot v^2 \cdot f, \tag{4.24}$$

where A is the number of switches per clock cycle, C is the total capacitance load, v is the supply voltage and f is the frequency of the machine. The energy consumed per machine i for the computation of task j is defined as the result of the following integration:

$$E_{ji} = \int_0^{completion[j][i]} Pow_{ji}(t)dt, \tag{4.25}$$

where $completion[j][i]$ is a completion time of the task j on machine i.

Based on Equation (4.23) the energy used for finishing the task j on machine i at level s_l can be defined as follows:

$$E_{ji}(s_l) = \gamma \cdot (f_{s_l}(i))_j \cdot f \cdot [(v_{s_l}(i))_j]^2 \cdot \widehat{ETC}[j][i][s_l], \tag{4.26}$$

where $\gamma = A \cdot C$ is a constant parameter for a given machine class, $(v_{s_l}(i))_j$ is a voltage supply value for class s^i and machine i at level s_l for computing task j, and $(f_{s_l}(i))_j$ is a corresponding relative frequency for machine i.

Therefore the computational times for each possible pair (j,i) at the level s_l can be calculated as follows:

$$\begin{aligned} Tim_{\{j,i,s_l\}} &= \gamma \cdot (f_{s_l}(i))_j \cdot f \cdot [(v_{s_l}(i))_j]^2 \cdot (f_{s_1}(i))_j \cdot ETC[j][i] = \\ &= \gamma \cdot f \cdot [(v_{s_l}(i))_j]^2 \cdot ETC[j][i] \end{aligned} \tag{4.27}$$

The cumulative energy utilized by the machine i for the completion of all tasks from the batch that are assigned to this machine, is defined in the following way:

$$\begin{aligned} E_i = \sum_{\substack{j \in Tasks(i) \\ l \in \hat{L}_j}} \{Tim_{\{j,i,s_l\}}\} + \gamma \cdot f \cdot [v_{s_{max}}]^2 \cdot ready_i + \gamma \cdot f_{s_{min}}(i) \cdot f \cdot \\ \cdot [v_{s_{min}}(i)]^2 \cdot Idle[i] = \gamma \cdot f \cdot \sum_{\substack{j \in Tasks(i) \\ l \in \hat{L}_i}} ([(v_{s_l}(i))_j]^2 \cdot ETC[j][i]) + \\ + [v_{s_{max}}(i)]^2 \cdot ready_i + f_{s_{min}}(i) \cdot [v_{s_{min}}(i)]^2 \cdot Idle[i] \end{aligned} \tag{4.28}$$

where $Idle[i]$ denotes an idle time of machine i, and $\hat{L}_i$ denotes a subset of DVFS levels used for the tasks assigned to machine i[3].

Finally, an average cumulative energy utilized by the grid system for completion of all tasks in the batch is defined as:

[3] All additional machine frequency transition overheads do not bear down on the overall ETC model with an active "energetic" module and are ignored.

$$E_{batch} = \frac{\sum_{i=1}^{m} E_i}{m} \tag{4.29}$$

This model is used in the following section for specification of two "energetic" scheduling scenarios.

4.4.3.1 Energy-aware Scheduling Scenarios

It is assumed machines supplied with DVFS modules can work in two following modes:

I. **Max-Min Mode**, where each machine works at the **maximal** DVFS level during the execution and computation of tasks and switch into idle mode after the execution of all tasks assigned to this machine;
II. **Modular Power Supply Mode**, where each machine can work at **different** DVFS levels during the task executions and can then enter into idle mode.

The procedures for calculation and optimization *Makespan*, *Flowtime* and cumulative energy utilized by the system, are different in the aforementioned scheduling scenarios and also additionally in secure and risky modes.

In **Max-Min Mode** the formulas for the completion time, *Makespan* and *Flowtime* are the same as in Eqs. (4.16), (4.17) and (4.18) in the risky mode; and Eqs. (4.13), (4.14) and (4.15) in the secure mode.

The idle time for machine i working in **Max-Min Mode** can be expressed as the difference between the *Makespan* and the completion time of this machine, that is to say:

$$Idle^r[i] = C^r_{max} - completion^r[i] \tag{4.30}$$

in the risky mode, and

$$Idle^s[i] = C^s_{max} - completion^s[i] \tag{4.31}$$

in the secure mode.

In the case of the machine with the maximal completion time (*Makespan*), the idle factor is zero.

In **Modular Power Supply Mode**, for each task-machine pair, the DSV level s_l must be specified. The formulas for computing the completion times in secure and risky scenarios at the level s^i can be defined as follows:

$$completion^s_{II}[i] = ready_i + \sum_{j \in Tasks(i)} \frac{1}{f_{s_l}(i)} \cdot \left(1 + P_f[j][i]\right) \cdot ETC[j][i]. \tag{4.32}$$

in the secure mode, and

$$completion^r_{II}[i] = completion_{II}[i] + completion^s_{res,II}[i] \tag{4.33}$$

in the risky mode, where $completion^s_{res,II}[i]$ is the completion time calculated for rescheduled tasks on machine i, and $completion_{II}[i]$ is calculated as follows:

$$completion_{II}[i] = ready_i + \sum_{j \in Tasks(i)} \frac{1}{f_{s_l}(i)} \cdot ETC[j][i]. \tag{4.34}$$

These formulas are using for the specification of the *Makespan* $\left(C^r_{max}\right)_{II}$ and $\left(C^s_{max}\right)_{II}$ in both security and risky modes. The *Flowtime* can be calculated from the modified Eq. (4.18) and (4.15) by replacing the $ECT[j][i]$ components by $\frac{1}{f_{s_l}(i)} \cdot ETC[j][i]$.

Finally, the formulas for idle times can be expressed as follows:

$$Idle^s_{II}[i] = \left(C^s_{max}\right)_{II} - completion^s_{II}[i] \tag{4.35}$$

$$Idle^r_{II}[i] = \left(C^r_{max}\right)_{II} - completion^r_{II}[i] \tag{4.36}$$

The average energy consumed in the system in **Min-Max Mode** and risky scenario is defined as follows:

$$\begin{aligned} E^r_I &= \tfrac{1}{m} \cdot \textstyle\sum_{i=1}^{m} \gamma \cdot completion^r[i] \cdot f \cdot [v_{s_{max}}(i)]^2 + \\ &+ \tfrac{1}{m} \cdot \textstyle\sum_{i=1}^{m} \gamma \cdot f_{s_{min}}(i) \cdot [v_{s_{min}}(i)]^2 \cdot Idle^r[i] \end{aligned} \tag{4.37}$$

Similar formula can be defined for the energy utilized in the secure mode E^s_I. It can be realized by replacing the $completion^r[i]$ and $Idle^r[i]$ in Eq. (4.37) by $completion^s[i]$ and $Idle^s[i]$ (see Eq. (4.13) and (4.31)).

In **Modular Power Supply Mode** the average cumulative energy is given by Eq. (4.29):

$$E_{II} = \frac{\sum_{i=1}^{m} E_i}{m} \tag{4.38}$$

where E_i is calculated by modification the Eq. (4.28) by using the Eq. (4.33)–(4.36).

The optimization procedure is two-steps. First the *Makespan* and *Flowtime* are minimized. Then, keeping the first two objectives at the optimal levels, the cumulative energy utilized in the system is also minimized.

4.5 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 a variety of fields [3], [2], [14], [29]. In grid scheduling these methods can tackle the various scheduling attributes and additional energy and security aspects.

The heuristic scheduling methods are usually classified into three main groups, namely (1) calculus-based (greedy algorithms and ad-hoc methods); (2) stochastic (guided and non-guided methods); and (3) enumerative methods (dynamic programming and branch-and-bound algorithm). In this work we focus on genetic-based methods that classified as population-based stochastic schedulers. Alg. 1 defines a

generic framework for genetic algorithm (GA) designed for grid scheduling. This framework is based on the general model of the conventional single-population GA dedicated [40].

Algorithm 1 A template of the genetic engine for *HGS-Sched*

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

The parameter e in Alg. 1 is the counter of the generations in GA (e counts the number of loops in GA). The populations in this algorithms are encoding by using the following two methodologies:

- ;
- **Direct Encoding:** A schedule vector S is expressed in the form:

$$S = [i_1, \ldots, i_n]^T, \tag{4.39}$$

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

- **Permutation-based Encoding:** A schedule vector is defined as follows:

$$Sch = [Sch_1, \ldots, Sch_n]^T, \tag{4.40}$$

where $Sch_i \in N$, $i = 1, \ldots, n$. Schedule Sch is the vector of labels of tasks assigned to the machines. For each machine the labels of the tasks assigned to this machine are sorted in ascending order by the completion times of the tasks.

In permutation-based representation some additional information about the numbers of tasks assigned to each machine is required. The total total number of tasks assigned to a machine i is denoted by $\widetilde{Sch_i}$ and is interpreted as the i-th coordinate of an assignment vector $\widetilde{Sch} = [\widetilde{Sch_1}, \ldots, \widetilde{Sch_m}]^T$, which defines in fact the loads of grid machines. The *direct representation* is used for encoding schedules in the base populations P^e and P^{e+1}, and *permutation representation* is used in P_c^e and P_m^t populations.

The initial population in Alg. 1 is generated by using the *Minimum Completion Time + Longest Job to Fastest Resource - Shortest Job to Fastest Resource 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 [51]. In LJFR-SJFR method initially the number of m tasks with the highest workload are assigned to the available m machines sorted in ascending order by the computing capacity criterion. Then the remaining unassigned tasks are allocated to the fastest available machines. In the MCT heuristics, a given task is assigned to the machine yielding the earliest completion time. The detailed definition of those procedures may be found in [11].

Alg. 1 was adapted to the CG scheduling problem through an implementation of specialized encoding methods and genetic operators. The operators from the following set were used in experiments presented in this book:

- **Selection operators:** *Linear Ranking*;
- **Crossover operators:** *Partially Mapped Crossover (PMX)* and *Cycle Crossover (CX)*;
- **Mutation operators:** *Move*, *Swap* and *Rebalancing*;
- **Replacement operators:** *Steady State*, *Elitist Generational*, *Struggle*.

All the above mentioned operators are commonly used in the genetic metaheuristics dedicated to solving combinatorial optimization problems [12]. The detailed definition and examples may be found in [5].

In *Linear Ranking* method a selection probability for each individual in a population is proportional to the rank of the individual. The rank of the worst individual is defined as zero, while the best rank is defined as $pop\_size - 1$, where pop_size is the size of the population.

The following crossover and mutation operators are implemented for permutation-based representation of the schedules. In *Partially Matched Crossover (PMX)* [15] a segment of one parent-chromosome is mapped to a segment of the other parent-chromosome (corresponding positions) and the remaining genes are exchanged according to the mapping 'relationship' of tasks and machines specified by the concrete scheduling rules. Tn *Cycle Crossover (CX)* [42], first, a cycle of alleles is identified. The crossover operator leaves the cycles unchanged, while the remaining segments in the parental strings are exchanged.

In *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 jobs per machine. It is realized by the modification of two coordinates in the vector $\tilde{u}$ of the schedule code in permutation-based representation. The main idea of the *Rebalancing* method is to improve the solution (by rebalancing the machine loads) and then mutate it. In rebalancing procedure, first, the most overloaded machine is selected. Two tasks j and $\hat{j}$ are identified in the following way: j is assigned to another machine i', $\hat{j}$ is assigned to i and $ETC[j][i'] \leq ETC[\hat{j}][i]$. Then the assignments are interchanged for tasks j and $\hat{j}$. In *Swap* mutation the indexes of two selected tasks in the schedule representation are swapped.

A base population for a new GA loop in Alg. 1 may be defined by using the *Elitist generational* replacement method, where a new population contains two best solutions from the old base population and the rest are the newly generated offsprings.

In the *Steady State* replacement method, the set of the best offsprings (the number of elements in this set is fixed) replaces the worst solutions in the old base population. The main drawback of this methods is that it can lead to the premature convergence of the algorithms in some local solutions. The *Struggle* replacement mechanism can be an effective tool for avoiding too fast of a scheduler's convergence to the local optima. In such method, new generations of individuals are created by replacing a part of the population by the individuals most similar – 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* [36] for measuring the distances between schedules according to the following formula:

$$sim_e(S^1; S^2) = \sqrt{\sum_{j=1}^{n} \frac{(S^1[j] - S^[j])^2}{\sigma_P^2}} \tag{4.41}$$

where σ_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., [7], [18]). However, the computational cost can be very high, because of the need to calculate 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. The value of this key 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).

Eighteen GA variants with all possible combinations of these operators are defined in Table 4.2.

All these algorithms have been empirically evaluated by using the grid simulator. The results of this empirical analysis are presented in the next section.

4.6 Empirical Evaluation of Genetic Grid schedulers

In this section we present a simple empirical analysis of the efficiency of 18 implementations of single-population GA schedulers defined in the previous section in the minimization of the scheduling objective function specified in Sec. 4.4. The relative performance of all schedulers has been quantified with the following four metrics:

- *Makespan* – the dominant scheduling criterion which can be calculated in various way depending on the security and energetic criteria (see Sec. 4.4.3);
- *Mean_Flowtime* – mean *Flowtime* calculated as follows:

$$Mean\_Flowtime = \frac{C_{sum}}{m} \tag{4.42}$$

Table 4.2: Eighteen variants of single-population GA-based schedulers

Scheduler	Crossover method	Mutation method	Replacement method
GA-PMX-M-SS	Partially Matched (PMX)	Move	Steady State
GA-PMX-M-EG	Partially Matched (PMX)	Move	Elitist Generational
GA-PMX-M-ST	Partially Matched (PMX)	Move	Struggle
GA-PMX-S-SS	Partially Matched (PMX)	Swap	Steady State
GA-PMX-S-EG	Partially Matched (PMX)	Swap	Elitist Generational
GA-PMX-S-ST	Partially Matched (PMX)	Swap	Struggle
GA-PMX-R-SS	Partially Matched (PMX)	Rebalancing	Steady State
GA-PMX-R-EG	Partially Matched (PMX)	Rebalancing	Elitist Generational
GA-PMX-R-ST	Partially Matched (PMX)	Rebalancing	Struggle
GA-CX-M-SS	Cycle (CX)	Move	Steady State
GA-CX-M-EG	Cycle (CX)	Move	Elitist Generational
GA-CX-M-ST	Cycle (CX)	Move	Struggle
GA-CX-S-SS	Cycle (CX)	Swap	Steady State
GA-CX-S-EG	Cycle (CX)	Swap	Elitist Generational
GA-CX-S-ST	Cycle (CX)	Swap	Struggle
GA-CX-R-SS	Cycle (CX)	Rebalancing	Steady State
GA-CX-R-EG	Cycle (CX)	Rebalancing	Elitist Generational
GA-CX-R-ST	Cycle (CX)	Rebalancing	Struggle

where C_{sum} similar to $Makespan$, can be calculated in various way depending on the security and energetic criteria (see Sec. 4.4.3);

- a relative energy consumption improvement rate expressed as follows:

$$Im(E) = \frac{E_I - E_{II}}{E_{batch}} \cdot 100\%, \tag{4.43}$$

where E_{II} and E_I are defined in Eq. (4.29) and Eq. (4.37) respectively;

- $FailureRate\ Fail_r$ parameter defined as follows:

$$Fail_r = \frac{n_{failed}}{n} \cdot 100\% \tag{4.44}$$

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

For providing the experiments and for simulating the grid environment we used the *Sim-G-Batch* grid simulator [28], that is based on the discrete event-based model *HyperSim-G* [52], and facilitates the evaluation of different scheduling heuristics under a variety of scheduling criteria across several grid scenarios. These scenarios are defined by the configuration of security conditions for scheduling and the access to the grid resources, grid size, energy utilization parameters, and system dynamics. The simulator allows the flexible activation or deactivation of all of the scheduling criteria and modules, as well as works with a mixture of meta-heuristic schedulers.

The main concept and general flow of the energy-aware and security version of the *Sim-G-Batch* simulator is presented in Fig. 4.3.

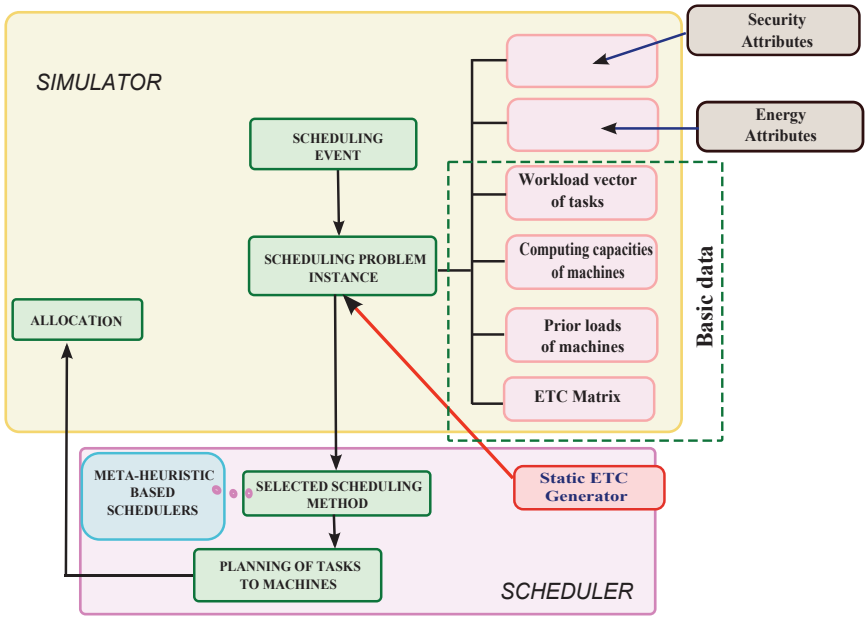

Fig. 4.3: General Flowchart of Security- and Energy-Aware *Sim-G-Batch* Simulator Linked to Scheduling

The basic set of the input data for the simulator includes:

- the workload vector of tasks,
- the computing capacity vector of machines,
- the vector of prior loads of machines, and
- the ETC matrix of estimated execution times of tasks on machines.

This set is extended by SD and Tl vectors (see Sec. 4.2) for the specification of the main security parameters and the following "energy" attributes:

- machine categories specification parameters (number of classes, maximal computational capacity value, computational capacity ranges interval for each class, machine operational speed parameter for each class, etc.);
- DVFS levels matrix for machine categories.

The simulator is highly parametrized in order to reflect numerous realistic grid scenarios. All schedulers are decoupled from the simulator main body and are implemented as the external libraries. The performance of all considered GA-based meta-heuristics has been analyzed in two grid size scenarios: Medium with 64 hosts and 1024 tasks, and Very Large with 256 hosts and 4096 tasks. The capacity of the resources and the workload of tasks are randomly generated by Gaussian distributions.

The sample values of key input parameters used in the experiments for the simulator are presented in Table 4.3[4].

Table 4.3: Values of key parameters of the grid simulator in static and dynamic cases

	Medium	**Very Large**
Static case		
Nb. of hosts	64	256
Resource cap.	$N(5000,875)$	
Total nb. of tasks	1024	4096
Workload of tasks	$N(250000000,43750000)$	
Dynamic case		
Init. hosts	64	256
Max. hosts	70	264
Min. hosts	50	240
Resource cap.	$N(5000,875)$	
Add host	$N(562500,84375)$	$N(437500,65625)$
Delete host	$N(625000,93750)$	
Init. tasks	768	3072
Total tasks	1024	4096
Workload	$N(250000000,43750000)$	
Both cases		
Security demands sd_j	$U[0.6;0.9]$	
Trust levels tl_i	$U[0.3;1]$	
Failure coefficient α	3	

The machines can work at 16 DVFS levels and can be categorized into three "energetic" resource classes, Class I, Class II, and Class III. The class identifiers have been selected randomly for the machines. The values of supply voltages and relative machine frequencies at all DVFS levels are specified in Table 4.1. The key parameters for the genetic schedulers are presented in Table 4.4.

[4] The following notation $U(a,b)$ and $N(a,b)$ is used for uniform and Gaussian probability distributions, respectively.

Table 4.4: GA setting for large static and dynamic benchmarks

Parameter	Elitist Generational/Struggle	Steady State
degree of branches (t)	0	
period_of_metaepoch (α)	$1/2 * n/10$	$(2 * n/10$
nb_of_metaepochs	10	
population size (pop_size)	$\lceil(\log_2 n)^2 - \log_2 n\rceil$	$4 * (\log_2 n - 1)$
intermediate pop.	$pop\_size - 2$	$(pop\_size)/3$
cross probab.	0.8	1.0
mutation probab.	0.2	
max_time_to_spend	40 sec. (*static*) / 75 sec. (*dynamic*)	

4.6.1 Results

Each experiment was repeated 30 times under the same configuration of operators and parameters. The results for *Makespan*, *Flowtime*, failure rate and relative energy consumption improvement rate averaged in 30 runs of the simulator are presented in Tables 4.5–4.11.

It follows from the results that the quality of scheduling strongly depends on the proper combination of crossover and mutation operations. In all considered instances the *PMX* crossover together with *Move* mutation give the worst results and the combination of *CX* crossover with *Rebalancing* mutation seems to be the most effective in the most of the instances. Indeed, in the static case $GA-CX-R-ST$ algorithm ranks first in about 75% of instances and $GA-CX-R-SS$ algorithm is the best in the remaining 25%of the total instances. In the dynamic scenario the situation is similar: $GA-CX-R-ST$ algorithm achieves the best results in about 62.5% of instances and $GA-CX-R-SS$ – in the remaining 27.5%. The $GA-CX-R-EG$ algorithm ranks in all cases as the second or third best scheduler. I can be observed that in the groups of algorithm with the same mutation and crossover operators, the *Struggle* replacement mechanism has the best positive impact on the algorithm performance. The average failure rates for the considered algorithms are in range 37% – 54%, and the energy improvement rate is in range 8% – 36%.

By summarizing the results for all the objective functions, the *GA-CX-R-ST* algorithm is the best in the minimization of all considered objectives. It should be also observed that in the risky mode almost all of the achieved results are worse than in the secure mode. It means that in the case of ignoring all security conditions the high failure rates increase the energy utilized by the system, and of course extend the deadlines for completing the particular tasks and the whole schedules.

Table 4.5: Average *Makespan* values for eighteen GA-based schedulers in **secure** scenario [±*s.d.*], (*s.d.* = standard deviation)

	Static		Dynamic	
Strategy	**Medium**	**Very Large**	**Medium**	**Very Large**
GA-PMX-M-SS	4165583.135 [± 690668.957]	4288022.184 [± 824291.171]	4294782.139 [± 911521.151]	4466888.391 [± 560336.505]
GA-PMX-M-EG	4177553.793 [± 190066.130]	4301234.468 [± 451450.111]	4279562.342 [± 375819.231]	4485562.553 [± 750553.639]
GA-PMX-M-ST	4157425.782 [± 379356.512]	4295435.433 [± 515425.756]	4278653.544 [± 489524.252]	4435334.722 [± 364469.977]
GA-PMX-S-SS	4126691.373 [± 7592355.678]	4268733.837 [± 575235.535]	4202645.677 [± 5733466.268]	4418525.678 [± 953456.457]
GA-PMX-S-EG	4148405.275 [± 43761.015]	4285671.797 [± 605962.237]	4216301.090 [± 53486.153]	4421427.663 [± 701992.116]
GA-PMX-S-ST	4133822.183 [± 380836.642]	4261593.241 [± 634705.248]	4199261.431 [± 555304.633]	439189.653 [± 711735.974]
GA-PMX-R-SS	4099722.422 [± 939509.617]	4209834.534 [± 505720.705]	4122903.467 [± 149437.882]	4332736.028 [± 563164.075]
GA-PMX-R-EG	4111018.744 [± 837452.949]	4217551.633 [± 540610.017]	4161359.893 [± 590881.527]	4375643.075 [± 748754.806]
GA-PMX-R-ST	4096915.832 [± 653833.933]	4202740.834 [± 468878.756]	4109927.347 [± 997874.784]	4313319.834 [± 752973.821]
GA-CX-M-SS	4057635.783 [± 711365.726]	4175408.673 [± 796525.362]	4034538.827 [± 933600.267]	4284402.632 [± 684131.232]
GA-CX-M-EG	4067320.534 [± 654607.977]	4184339.256 [± 582147.023]	4090546.734 [± 751364.567]	4303769.235 [± 952170.387]
GA-CX-M-ST	404176222.362 [± 987536.453]	4167744.674 [± 803122.735]	4009105.874 [± 679674.456]	4241630.356 [± 728082.735]
GA-CX-S-SS	3954447.634 [± 886725.393]	4125317.764 [± 671196.560]	39872916.546 [± 998157.753]	4205709.634 [± 564696.828]
GA-CX-S-EG	4004965.356 [± 918998.520]	4156258.356 [± 713229.061]	3990734.764 [± 594713.746]	4222631.465 [± 957740.279]
GA-CX-S-ST	3940265.563 [± 989958.718]	4098563.182 [± 455636.448]	41614537.863 [± 56051.579]	4219725.327 [± 988429.144]
GA-CX-R-SS	392372.215 [± 384195.783]	405397.326 [± 697330.373]	3985763.378 [± 581510.863]	4152623.367 [± 712805.735]
GA-CX-R-EG	3912183.536 [± 341357.564]	4095293.987 [± 981563.633]	4093277.356 [± 754356.222]	4197426.546 [± 774433.987]
GA-CX-R-ST	**3907233.453 [± 552722.245]**	**3982865.453 [± 827733.653]**	**3900433.453 [± 368863.563]**	**4100641.722 [± 704973.453]**

Table 4.6: Average *Makespan* values for eighteen GA-based schedulers in **risky** scenario [±*s.d.*], (*s.d.* = standard deviation)

	Static		Dynamic	
Strategy	**Medium**	**Very Large**	**Medium**	**Very Large**
GA-PMX-M-SS	4183064.456 [± 303594.681]	41996544.866 [± 633541.835]	4272342.367 [± 523861.891]	4351579.387 [± 955889.372]
GA-PMX-M-EG	4196660.637 [± 795391.867]	4207559.346 [± 673312.285]	4250769.729 [± 570063.523]	4378943.912 [± 663276.562]
GA-PMX-M-ST	41666682.922 [± 557522.757]	4182062.259 [± 659041.716]	4242506.997 [± 564267.992]	4342520.961 [± 435640.377]
GA-PMX-S-SS	4150255.659 [± 364204.730]	4156125.373 [± 612692.293]	4239582.165 [± 566626.175]	4323745.969 [± 685834.282]
GA-PMX-S-EG	4149105.038 [± 440578.157]	416379.853 [± 472235.032]	4220166.253 [± 837511.681]	4309346.443 [± 515978.598]
GA-PMX-S-ST	4133090.502 [± 271571.051]	4177107.429 [± 473557.935]	4198308.287 [± 492853.075]	4294855.678 [± 430583.789]
GA-PMX-R-SS	4126676.780 [± 728966.192]	4118808.678 [± 649630.565]	4164756.208 [± 56766.709]	4254796.820 [± 580607.466]
GA-PMX-R-EG	4129602.691 [± 270662.077]	4109778.902 [± 913428.430]	4182790.083 [± 761523.938]	4270973.780 [± 100219.228]
GA-PMX-R-ST	4063022.741 [± 430715.830]	4077928.331 [± 410172.944]	4158207.631 [± 611305.330]	4232768.084 [± 499217.899]
GA-CX-M-SS	4096232.073 [± 554669.949]	4093218.110 [± 602133.299]	4114422.323 [± 721311.202]	4188744.263 [± 672746.995]
GA-CX-M-EG	4109712.181 [± 869717.384]	4099788.995 [± 453095.636]	4141965.814 [± 817008.729]	4209289.242 [± 636853.293]
GA-CX-M-ST	4054429.908 [± 464589.755]	4070157.151 [± 446428.587]	4127500.870 [± 870899.646]	4198327.481 [± 631520.079]
GA-CX-S-SS	4022572.196 [± 516330.313]	4032408.549 [± 590815.709]	4090392.967 [± 898536.995]	4129281.695 [± 917216.254]
GA-CX-S-EG	4015442.255 [± 405905.662]	4033181.392 [± 650601.916]	4106452.250 [± 982489.800]	4164853.468 [± 785931.696]
GA-CX-S-ST	4006934.189 [± 436482.946]	4010678.953 [± 516992.488]	4039201.986 [± 816514.067]	4140525.379 [± 633513.047]
GA-CX-R-SS	3986872.198 [± 806632.171]	3972483.354 [± 618035.814]	4024651.986 [± 436431.848]	4109893.365 [± 747400.818]
GA-CX-R-EG	3999111.928 [± 692024.001]	4002987.835 [± 467779.063]	4089790.735 [± 632965.292]	4091857.736 [± 791598.934]
GA-CX-R-ST	**3955725.386 [± 911937.937]**	**39677356.846 [± 879753.092]**	4030546.634 [± 850836.928]	**4038634.546 [± 932835.453]**

Table 4.7: Average *Flowtime* values for eighteen GA-based schedulers in **secure** scenario [±*s.d.*], (*s.d.* = standard deviation)

	Static		Dynamic	
Strategy	**Medium**	**Very Large**	**Medium**	**Very Large**
GA-PMX-M-SS	2193567211.836 [± 128370293.469]	8309597733.826 [± 291748525.988]	2399334723.236 [± 411586354.582]	8401592320.927 [± 889213844.016]
GA-PMX-M-EG	2206734272.356 [± 135504556.429]	8326428289.314 [± 13744410.910]	2404322051.418 [± 439654339.019]	8410389312.587 [± 989122999.052]
GA-PMX-M-ST	2182232499.779 [± 218224825.514]	8297351808.190 [± 207359074.643]	2357752153.775 [± 476389087.741]	8395922739.597 [± 606291276.896]
GA-PMX-S-SS	2158140985.025 [± 180302378.918]	8261813008.415 [± 240332696.510]	2316860418.277 [± 593778909.154]	8367766411.152 [± 755794904.240]
GA-PMX-S-EG	2175542598.454 [± 162987312.683]	828568131.930 [± 369229026.877]	2332618330.048 [± 45367872.253]	8388110055.663 [± 507151151.344]
GA-PMX-S-ST	2147247760.724 [± 211590465.457]	8271025623.997 [± 453137510.918]	2320515589.851 [± 711100452.764]	8373206007.045 [± 784801239.205]
GA-PMX-R-SS	21142225326.145 [± 240067553.885]	8249643747.642 [± 253050662.960]	2286055243.299 [± 577523722.350]	8338732539.636 [± 741000998.450]
GA-PMX-R-EG	2136071183.723 [± 143892082.090]	8258614783.371 [± 233840747.951]	2292713481.576 [± 377833946.994]	835216627.023 [± 699126484.932]
GA-PMX-R-ST	2117878614.800 [± 196255094.628]	8227564670.521 [± 225294411.337]	2283126157.824 [± 543390178.534]	8322161405.019 [± 736033399.656]
GA-CX-M-SS	2105676446.564 [± 18596758.737]	8195359732.256 [± 282655389.940]	2263563557.141 [± 539894530.830]	8294397268.110 [± 790327708.149]
GA-CX-M-EG	2125655077.793 [± 142293067.762]	8238431289.893 [± 212224990.911]	2274715011.673 [± 604531703.308]	8306547838.652 [± 812904982.726]
GA-CX-M-ST	21111848829.461 [± 110335889.154]	8204221008.299 [± 254605827.121]	2238467253.243 [± 513415557.850]	8272449832.295 [± 822664629.425]
GA-CX-S-SS	2084338418.886 [± 165598016.948]	8179625719.400 [± 250682728.242]	2204403080.180 [± 430849430.300]	8242964769.102 [± 600614818.712]
GA-CX-S-EG	2097868914.479 [± 17614361.064]	8190763974.690 [± 334857074.927]	2227164271.644 [± 473657175.059]	8260315502.581 [± 569231789.391]
GA-CX-S-ST	2078132035.030 [± 120473769.993]	8166781141.473 [± 250660461.860]	2195675260.221 [± 577482551.646]	8253290991.790 [± 769767684.495]
GA-CX-R-SS	2060575340.426 [± 123314394.677]	8138208217.698 [± 772885985.554]	**2177096342.984 [± 489373983.094]**	**8211987409.256 [± 999265736.534]**
GA-CX-R-EG	2066071183.009 [± 103578848.015]	8143852396.768 [± 272146853.672]	2196929745.169 [± 533701714.987]	8235408309.560 [± 795752579.317]
GA-CX-R-ST	**2035128734.875 [± 55234984.937]**	**8109087253.984 [± 339791854.937]**	2199758911.356 [± 436450229.234]	8221050176.985 [± 708270871.387]

Table 4.8: Average *Flowtime* values for eighteen GA-based schedulers in **risky** scenario [±*s.d.*], (*s.d.* = standard deviation)

	Static		Dynamic	
Strategy	**Medium**	**Very Large**	**Medium**	**Very Large**
GA-PMX-M-SS	2343724728.245 [± 598538835.360]	8321846250.347 [± 431346593.814]	2413245472.632 [± 417986755.794]	8644534678.245 [± 404482983.284]
GA-PMX-M-EG	2389239424.349 [± 129097532.581]	8347268532.324 [± 432955978.981]	2436061767.548 [± 701148099.631]	8660435684.376 [± 664054585.165]
GA-PMX-M-ST	2307355142.051 [± 126906586.696]	8308727439.878 [± 256156869.233]	2403766189.879 [± 469026059.421]	8605175251.450 [± 693219859.321]
GA-PMX-S-SS	2263649999.867 [± 268317752.960]	8262608448.916 [± 216787358.248]	2360160544.425 [± 656197729.295]	8502620979.464 [± 851502055.485]
GA-PMX-S-EG	2286784630.430 [± 184783757.268]	8292058041.397 [± 499461667.383]	2393131389.346 [± 454805534.425]	8531596508.841 [± 573812983.895]
GA-PMX-S-ST	2237247760.724 [± 244213856.135]	8271025623.997 [± 434633012.674]	2377515589.851 [± 788991696.341]	8513206007.045 [± 719513808.715]
GA-PMX-R-SS	2226429310.580 [± 255806730.308]	8229357685.290 [± 290693183.985]	2333974209.902 [± 519849125.536]	8472673073.563 [± 743945636.397]
GA-PMX-R-EG	2213613356.567 [± 167571940.967]	8254728732.731 [± 283645546.676]	2348829866.347 [± 526061100.837]	8493687750.198 [± 784339554.900]
GA-PMX-R-ST	2151930817.794 [± 134663685.381]	8184860096.988 [± 242728273.618]	2320407124.381 [± 528711756.114]	8466099314.428 [± 728490457.382]
GA-CX-M-SS	2188284638.934 [± 162112136.850]	8199577483.835 [± 257276315.811]	2239455642.778 [± 387064110.517]	8426829568.357 [± 537095310.247]
GA-CX-M-EG	2193593276.050 [± 163978670.974]	8211445673.176 [± 173050142.794]	2296573568.612 [± 621630992.871]	8457381627.647 [± 596961998.050]
GA-CX-M-ST	2132978230.586 [± 129547599.824]	8166440897.331 [± 182827503.146]	2270646610.327 [± 616490962.213]	8432573052.527 [± 885691236.077]
GA-CX-S-SS	2126523158.454 [± 228154433.283]	8161517451.690 [± 277432894.907]	2222289600.939 [± 552469744.108]	8365526025.239 [± 773215992.243]
GA-CX-S-EG	2113772162.066 [± 187159613.747]	8154839647.617 [± 326230384.931]	2233669538.605 [± 620551577.626]	8392873916.020 [± 859689519.119]
GA-CX-S-ST	2106340445.440 [± 138247946.733]	8146356979.846 [± 239873730.726]	2218990182.691 [± 525735729.121]	8345528446.458 [± 947608913.414]
GA-CX-R-SS	2076534164.177 [± 181982147.103]	**81127123653.774 [± 19799961.957]**	221654378.495 [± 48988254.993]	**8291082340.932 [± 834522826.826]**
GA-CX-R-EG	2098943746.287 [± 143403467.472]	8138965387.563 [± 176627923.813]	2208567534.205 [± 564619337.921]	8339386800.606 [± 730340037.273]
GA-CX-R-ST	**2020734590.928 [± 2393872683.029]**	8122230062.891 [± 217213422.096]	**2168923672.647 [± 8893693282.361]**	8310330051.728 [± 711071173.232]

Table 4.9: Average $Fail_r$ values for eighteen GA-based schedulers in **secure** scenario [$\pm s.d.$], ($s.d.$ = standard deviation)

	Static		Dynamic	
Strategy	**Medium**	**Very Large**	**Medium**	**Very Large**
GA-PMX-M-SS	44.67 [± 7.9]	49.40 [± 8.9]	43.81 [± 8.12]	47.12 [± 6.05]
GA-PMX-M-EG	47.83 [± 9.82]	47.01 [± 5.54]	44.80 [± 8.19]	48.63 [± 7.50]
GA-PMX-M-ST	43.547 [± 7.91]	47.95 [± 7.56]	40.77 [± 5.22]	48.37 [± 6.46]
GA-PMX-S-SS	42.99 [± 9.92]	48.68 [± 6.75]	44.04 [± 9.79]	47.22 [± 8.56]
GA-PMX-S-EG	43.48 [± 7.61]	49.67 [± 6.09]	44.16 [± 8.53]	47.66 [± 7.99]
GA-PMX-S-ST	41.82 [± 7.98]	48.93 [± 9.23]	43.26 [± 5.20]	49.11 [± 7.73]
GA-PMX-R-SS	43.09 [± 9.37]	48.55 [± 5.70]	43.90 [± 9.43]	48.74 [± 5.65]
GA-PMX-R-EG	43.07 [± 8.37]	46.47 [± 5.41]	43.69 [± 5.92]	48.93 [± 7.48]
GA-PMX-R-ST	42.96 [± 5.53]	47.39 [± 6.88]	43.06 [± 9.72]	48.74 [± 5.52]
GA-CX-M-SS	42.78 [± 7.15]	47.93 [± 7.25]	42.53 [± 9.25]	48.10 [± 6.93]
GA-CX-M-EG	41.96 [± 6.54]	48.39 [± 8.22]	42.94 [± 7.3]	49.11 [± 9.57]
GA-CX-M-ST	41.64 [± 9.77]	47.91 [± 8.64]	42.12 [± 6.79]	48.41 [± 7.24]
GA-CX-S-SS	42.54 [± 5.66]	45.65 [± 7.71]	43.28 [± 6.15]	48.65 [± 9.69]
GA-CX-S-EG	42.84 [± 9.55]	44.26 [± 9.30]	40.45 [± 9.71]	48.22 [± 9.52]
GA-CX-S-ST	38.66 [± 9.59]	46.08 [± 7.58]	39.47 [± 6.5]	48.27 [± 8.98]
GA-CX-R-SS	39. 13 [± 8.57]	43.34 [± 9.78]	41.67 [± 8.15]	46.12 [± 7.18]
GA-CX-R-EG	39.72 [± 6.56]	43.75 [± 6.77]	41.03 [± 9.46]	44.97 [± 7.84]
GA-CX-R-ST	**37.24 [± 4.27]**	**42.20 [± 6.92]**	**38.33 [± 9.88]**	**43.05 [± 7.88]**

Table 4.10: Average $Fail_r$ values for eighteen GA-based schedulers in **risky** scenario [$\pm s.d.$], ($s.d.$ = standard deviation)

	Static		Dynamic	
Strategy	**Medium**	**Very Large**	**Medium**	**Very Large**
GA-PMX-M-SS	44.29 [± 6.82]	51.99 [± 6.43]	45.66 [± 5.29]	53.65 [± 9.52]
GA-PMX-M-EG	44.68 [± 7.95]	52.07 [± 6.68]	45.50 [± 6.02]	53.72 [± 9.92]
GA-PMX-M-ST	44.92 [± 7.82]	51.80 [± 6.83]	45.49 [± 9.94]	53.42 [± 4.35]
GA-PMX-S-SS	44.02 [± 9.43]	51.88 [± 7.39]	45.39 [± 6.68]	53.21 [± 6.83]
GA-PMX-S-EG	43.28 [± 10.02]	51.99 [± 7.37]	45.12 [± 8.93]	53.09 [± 8.87]
GA-PMX-S-ST	43.94 [± 8.21]	51.77 [± 5.36]	44.98 [± 5.83]	52.94 [± 7.66]
GA-PMX-R-SS	43.91 [± 8.99]	51.18 [± 5.84]	44.64 [± 5.67]	52.54 [± 5.87]
GA-PMX-R-EG	43.99 [± 6.29]	51.99 [± 9.34]	44.83 [± 7.69]	52.53 [± 10.02]
GA-PMX-R-ST	43.64 [± 8.55]	52.03 [± 9.64]	43.93 [± 9.34]	52.01 [± 5.38]
GA-CX-M-SS	44.01 [± 9.22]	51.93 [± 8.33]	43.14 [± 7.74]	51.88 [± 9.95]
GA-CX-M-EG	43.95 [± 8.69]	51.29 [± 7.36]	43.02 [± 8.17]	52.09 [± 5.73]
GA-CX-M-ST	42.99 [± 10.13]	50.92 [± 8.63]	42.85 [± 8.70]	51.98 [± 7.92]
GA-CX-S-SS	42.22 [± 5.98]	50.78 [± 5.83]	42.99 [± 6.89]	51.96 [± 9.67]
GA-CX-S-EG	41.97 [± 7.03]	49.63 [± 9.92]	42.93 [± 8.28]	51.62 [± 8.32]
GA-CX-S-ST	41.08 [± 7.55]	50.20 [± 10.22]	43.03 [± 9.53]	51.93 [± 8.76]
GA-CX-R-SS	41.85 [± 8.06]	49.83 [± 9.53]	42.73 [± 6.72]	51.01 [± 7.47]
GA-CX-R-EG	40.59 [± 6.92]	49.21 [± 9.02]	4240 [± 7.83]	50. 39 [± 9.93]
GA-CX-R-ST	**39.88 [± 9.04]**	**47.92 [± 8.88]**	41.92 [± 9.24]	**50.38 [± 7.98]**

Table 4.11: Average $Im(E)$ values for eighteen GA-based schedulers in **secure** scenario [$\pm s.d.$], ($s.d.$ = standard deviation)

	Static		Dynamic	
Strategy	**Medium**	**Very Large**	**Medium**	**Very Large**
GA-PMX-M-SS	17.65 [± 2.49]	13.95 [± 2.94]	15.99 [± 4.1]	12.01 [± 1.89]
GA-PMX-M-EG	17.99 [± 3.55]	15.24 [± 1.44]	16.44 [± 2.39]	13.10 [± 1.82]
GA-PMX-M-ST	18.10 [± 2.18]	15.37 [± 2.07]	16.35 [± 2.47]	12.95 [± 1.93]
GA-PMX-S-SS	18.03 [± 3.09]	15.61 [± 2.40]	16.31 [± 3.59]	13.06 [± 5.50]
GA-PMX-S-EG	18.73 [± 4.26]	15.85 [± 3.62]	16.32 [± 3.45]	13.80 [± 2.50]
GA-PMX-S-ST	19.67 [± 4.98]	16.10 [± 4.51]	16.25 [± 3.11]	13.73 [± 2.78]
GA-PMX-R-SS	19.65 [± 2.40]	16.49 [± 2.53]	16.86 [± 3.7]	13.38 [± 2.74]
GA-PMX-R-EG	19.56 [± 3.73]	16.36 [± 3.26]	17.28 [± 4.01]	15.11 [± 4.28]
GA-PMX-R-ST	19.76 [± 3.75]	18.63 [± 2.29]	19.04 [± 5.99	15.77 [± 3.74]
GA-CX-M-SS	20.564 [± 3.85]	19.53 [± 2.82]	18.63 [± 1.99]	15.94 [± 3.74]
GA-CX-M-EG	21.23 [± 5.89]	20.43 [± 2.12]	18.36 [± 6.04]	17.98 [± 3.12]
GA-CX-M-ST	23.84 [± 3.98]	22.12 [± 4.39]	18.37 [± 5.18]	16.28 [± 4.02]
GA-CX-S-SS	25.70 [± 6.33]	20.83 [± 7.80]	19.27 [± 4.85]	19.02 [± 6.00]
GA-CX-S-EG	27.72 [± 2.32]	21.90 [± 7.34]	20.82 [± 4.29]	19.99 [± 4.65]
GA-CX-S-ST	26.87 [± 6.83]	21.35 [± 5.25]	20.37 [± 5.77]	20.23 [± 7.06]
GA-CX-R-SS	26.99 [± 5.89]	21.02 [± 5.32]	22.87 [± 4.19]	20.39 [± 6.37]
GA-CX-R-EG	28.14 [± 4.22]	21.63 [± 5.30]	27.91 [± 8.01]	20.19 [± 5.83]
GA-CX-R-ST	**35.38 [± 5.83]**	**28.15 [± 3.79]**	**30.93 [± 4.92]**	**32.05 [± 5.28]**

Table 4.12: Average $Im(E)$ values for eighteen GA-based schedulers in **risky** scenario [$\pm s.d.$], ($s.d.$ = standard deviation)

	Static		Dynamic	
Strategy	**Medium**	**Very Large**	**Medium**	**Very Large**
GA-PMX-M-SS	10.43 [± 1.59]	9.98 [± 1.43]	8.12 [± 1.41]	8.06 [± 1.40]
GA-PMX-M-EG	11.13 [± 1.12]	10.23 [± 1.43]	8.42 [± 1.70]	8.66 [± 1.66]
GA-PMX-M-ST	11.07 [± 0.99]	10.08 [± 1.25]	8.34 [± 1.46]	8.60 [± 1.69]
GA-PMX-S-SS	13.63 [± 1.82]	12.62 [± 1.418]	11.60 [± 1.65]	11.05 [± 1.85]
GA-PMX-S-EG	14.86 [± 1.84]	12.92 [± 1.49]	11.93 [± 1.45]	11.31 [± 1.57]
GA-PMX-S-ST	15.37 [± 2.44]	13.71 [± 4.43]	12.37 [± 1.781]	11.325 [± 1.75]
GA-PMX-R-SS	15.96 [± 2.55]	14.29 [± 2.90]	14.33 [± 3.51]	13.47 [± 3.74]
GA-PMX-R-EG	16.95 [± 1.67]	15.44 [± 2.83]	15.28 [± 3.52]	14.03 [± 4.70]
GA-PMX-R-ST	18.41 [± 3.76]	17.83 [± 2.42]	15.34 [± 4.21]	15.08 [± 3.92]
GA-CX-M-SS	18.41 [± 2.62]	16.99 [± 2.571]	14.22 [± 3.82]	13.00 [± 2.78]
GA-CX-M-EG	18.25 [± 3.97]	17.04 [± 3.05	14.83 [± 2.21]	13.40 [± 3.59]
GA-CX-M-ST	19.03 [± 2.94]	17.01 [± 3.28]	15.01 [± 3.64]	14.21 [± 1.85]
GA-CX-S-SS	19.26 [± 2.78]	18.83 [± 2.77]	17.45 [± 5.58]	15.21 [± 4.73]
GA-CX-S-EG	19.74 [± 1.87]	18.36 [± 3.29]	18.91 [± 3.22]	16.57 [± 2.81]
GA-CX-S-ST	19.34 [± 2.13]	18.35 [± 2.39]	18.78 [± 3.52]	16.98 [± 3.94]
GA-CX-R-SS	20.66 [± 2.98]	18.98 [± 1.97]	19.96 [± 2.99]	17.91 [± 3.82]
GA-CX-R-EG	20.17 [± 3.65]	19.22 [± 4.29]	20.06 [± 3.02]	17.98 [± 5.00]
GA-CX-R-ST	**21.77 [± 2.48]**	**20.20 [± 2.2]**	**21.33 [± 3.52]**	**18.03 [± 3.71]**

4.7 Multi-population Genetic Grid Schedulers

Due to sheer size of the grid and the dynamics of the grid environment, the exploration of the optimization landscapes generated for the grid scheduling problems remains challenging task for simple genetic-based grid schedulers, especially in the case of many different scheduling criteria (not just *Makespan* and *Flowtime*). In such case the effectiveness of the algorithms may be improved by executing them simultaneously on many populations as parallel processes. All single-population GAs presented in Sec. 4.5 may be used as main genetic mechanisms in multi-population genetic meta-heuristics. In this work, we consider two following multi-population genetic-based risk resilient grid schedulers:

- ***HGS-Sched (R)*** – multi-population hierarchical *HGS-Sched* scheduler working in the risky mode;
- ***HGS-Sched (S)*** – multi-population hierarchical *HGS-Sched* scheduler working in the secure mode;
- ***IGA (R)*** – multi-population island GA scheduler working in the risky mode;
- ***IGA (S)*** – multi-population island GA scheduler working in the secure mode.

The main idea of the Hierarchic Genetic Scheduler (HGS-Sched) based on the concept of a concurrent search in grid environment by the execution of many dependent evolutionary processes. The strategy is modelled as a multi-level decision tree. The search process in *HGS-Sched* is initialized by activating a scheduler with low search accuracy. This scheduler is the main module of the entire strategy and is responsible for the "management" of the general structure of the tree[5] and exploration of new and unrecognized regions in the optimization domain. The accuracy of search in *HGS-Sched* branches is defined by the *degree* parameter, and depends on the mutation rate and in fact the size of the population, which can be different in the branches of different degrees.

Figure 4.4 depicts a simple graphical representation of 3-level structure of *HGS-Sched*.

New branches are sprouted by using the sprouting procedure SO after execution of α-generation evolutionary processes (α-*periodic metaepoch* Met_α). The extension of the tree is steered by using the *Branch Comparison* (BC). The detailed definition and interpretation of all *HGS-Sched* operators can be found in [29].

The *Island Genetic Algorithm (IGA)* [48] is a well-known parallel GA technique. An initial (usually large) population is divided into several sub-populations, "islands" or "demes", for which single-population GAs with identical configurations of the parameters and operators are activated (one algorithm for each deme). After a fixed number of iterations, denoted as it_d, the migration procedure is activated. It enables a partial exchange (usually according to the standard ring topology) of the individuals among islands. The relative amount of the migrating individuals, represented by *mig*, is the algorithm global parameter commonly known as the *migration rate*, and calculated as:

[5] The scheduler with the lowest accuracy of search is called *the core* of the HGS tree structure.

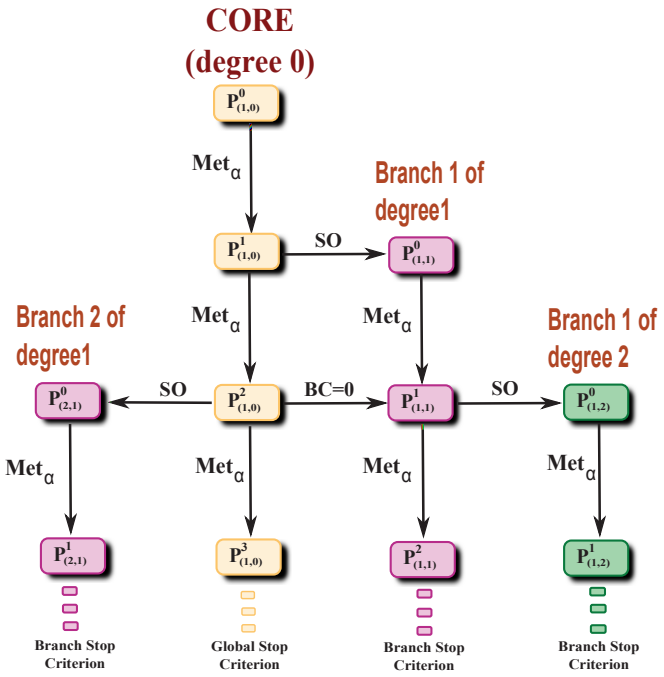

Fig. 4.4: 3 levels of *HGS-Sched* tree structure

$$mig = \frac{m_{deme}}{deme} \cdot 100\% \tag{4.45}$$

where *deme* is the size of the sub-population in *IGA* and m_{deme} is the number of migrating individuals in each deme. The procedure of migration is repeated after each execution of it_d iterations of genetic algorithm in each sub-population.

The $GA-CX-R-ST(R)$ and $GA-CX-R-ST(S)$ algorithms, selected as the most effective single-population schedulers in Sec. 4.6, have been used as the main genetic mechanisms in all considered implementation of *HGS-Sched* (in all types of branches) and *IGA*.

4.7.1 Empirical Analysis

In this section we present results of the empirical evaluation of two types of multi-population genetic schedulers and compare them with the results achieved by $GA-CX-R-ST(R)$ and $GA-CX-R-ST(S)$ algorithms. Similarly to the previous experiments, we have used the*Sim-G-Batch* grid simulator with the same configuration as specified in Sec. 4.6. The configurations of key parameters for both implementations of $GA-CX-R-ST(x)$ algorithms are the same as in the experiment

provided for all types of single-population schedulers (see Table 4.4), the parameters for *IGA* and *Green-HGS-Sched* meta-heuristics are presented in Tables 4.13 and 4.14.

Table 4.13: *HGS-Sched* settings for static and dynamic benchmarks

Parameter	
period_of_metaepoch	$20*n$
nb_of_metaepochs	10
degrees of branches (t)	0 and 1
population size in the core	$3*(\lceil 4*(\log_2 n-1)/(11.8)\rceil)$
population size in the sprouted branches (b_pop_size)	$(\lceil (4*(\log_2 n-1)/(11.8)\rceil)$
intermediate pop. in the core	$abs((r\_pop\_size)/3)$
intermediate pop. in the sprouted branch	$abs((b\_pop\_size)/3)$
cross probab.	0.9
mutation probab. in core	0.4
mutation probab. in the sprouted branches	0.2
max_time_to_spend	40 secs (*static*) / 70 secs (*dynamic*)

Table 4.14: Configuration of *IGA* algorithm

Parameter	
it_d	$20*n$
mig	5 %
number of islands (demes)	10
cross probab.	1.0
mutation probab.	0.2
max_time_to_spend	40 secs (*static*) / 70 secs (*dynamic*)

4.7.2 Results

Each experiment was repeated 30 times under the same configuration of operators and parameters. The box-plots of the statistical analysis of the values of the four scheduler performance measures averaged in 30 runs of the simulator are presented in Fig. 4.5–4.8.

In the case of $Makespan$, $Flowtime$ and $Fail_r$ the best results are achieved by the $HGS-Sched(S)$ algorithm. Especially in the dynamic grid the difference in the results generated by this scheduler and the the other meta-heuristics are significant. In the case of "Medium" grid the single-population $GA-CX-R-ST(S)$ algorithm is more effective in the $Makespan$ minimization that the secure implementation of the island model, which shows that in this case the best found solutions of the problem

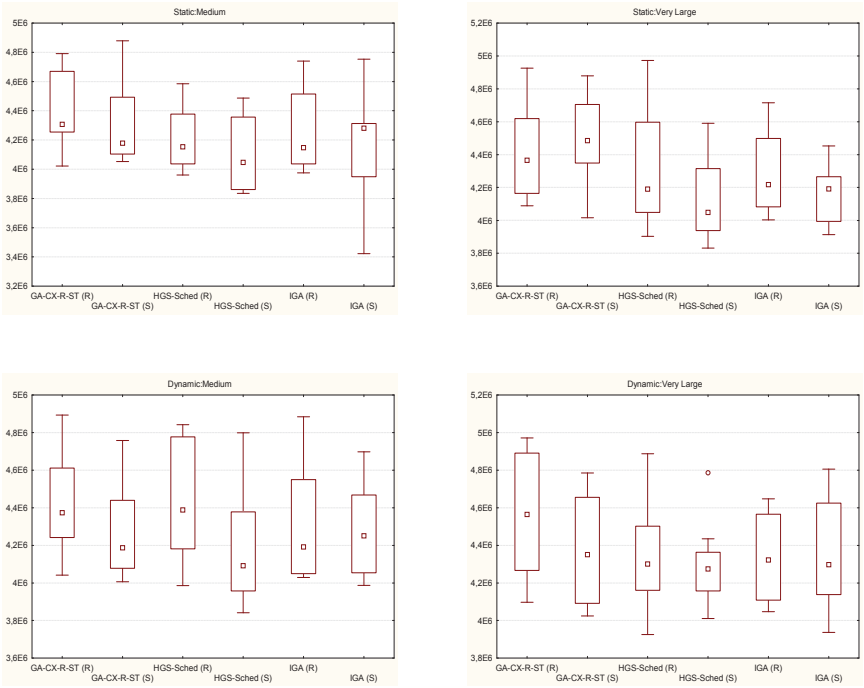

Fig. 4.5: The box-plot of the results for *Makespan* in static and dynamic scenarios

may be located very close to each other, and the island procedure do not improve the quality of search of the single-population scheduling mechanism implemented in each deme of *IGA* . In all of scheduling scenarios the secure implementations of the algorithms from the same class are more effective than their risky implementations for all three above mentioned scheduling criteria.

The situation is a bit different in the case of the minimization of the energy consumption. In this case the risky implementations of *HGS-Sched* and *IGA* algorithms achieved the best (highest) values of the energy improvement rates in the "Medium" and "Very large" grids respectively. This may leads to the conclusion that the lowering the power supply of the system in those cases didn't increase the *Makespan* and *Flowtime* values, which are rather big for those two schedulers, but allowed to reduce significantly the energy utilization.

It can be observed that *HGS-Sched (S)* is the most stable scheduler in all considered scenarios. The differences between the minimal and maximal values, and the first and the third quantiles are the lowest for this meta-heuristic.

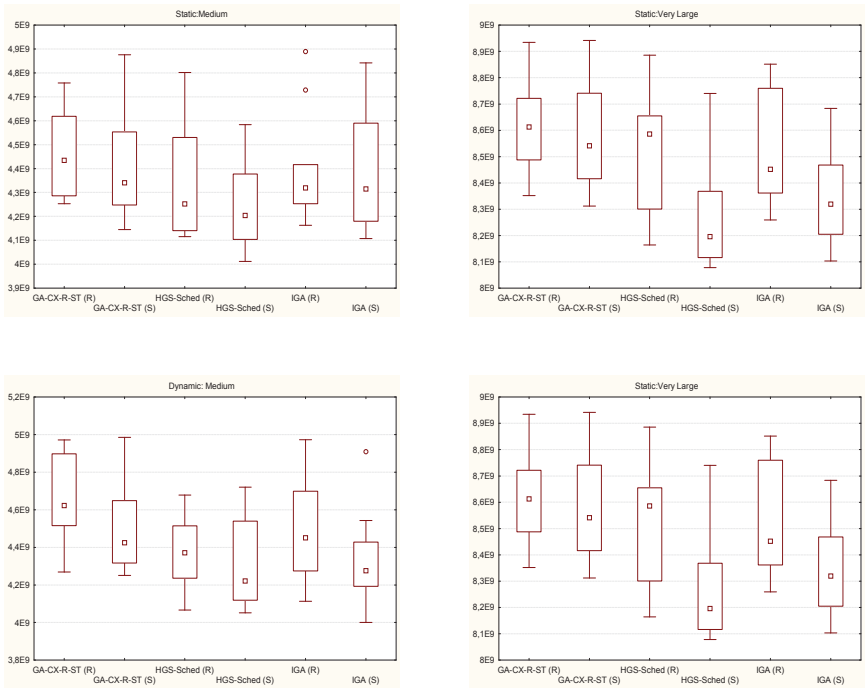

Fig. 4.6: The box-plot of the results for *Mean_Flowtime* in static and dynamic scenarios

4.8 Related Work

The security and resource reliability issues have been intensively studied in numerous publications over the last years [41]. However, many well-known scheduling approaches for grid computing largely ignore the security factor, with only a handful of exceptions.

A lot of efforts have been made on developing the intelligent management modules in the grid systems for controlling the accessing services through the authentication [20] in online scheduling, or for monitoring the execution of the grid applications and detection the resource failures due to the security restrictions [21]. In [1] 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.

Although the security-aware scheduling in grids is well studied, there are not so many examples of successful application of the genetic-based meta-heuristics for solving this problem. In most of the publications in the domain security and task abortion mechanisms are usually implemented as the external procedures separated from the core of the scheduling system. In [47] the authors defines the security as additional scheduling criterion in online grid scheduling and aggregated this criterion

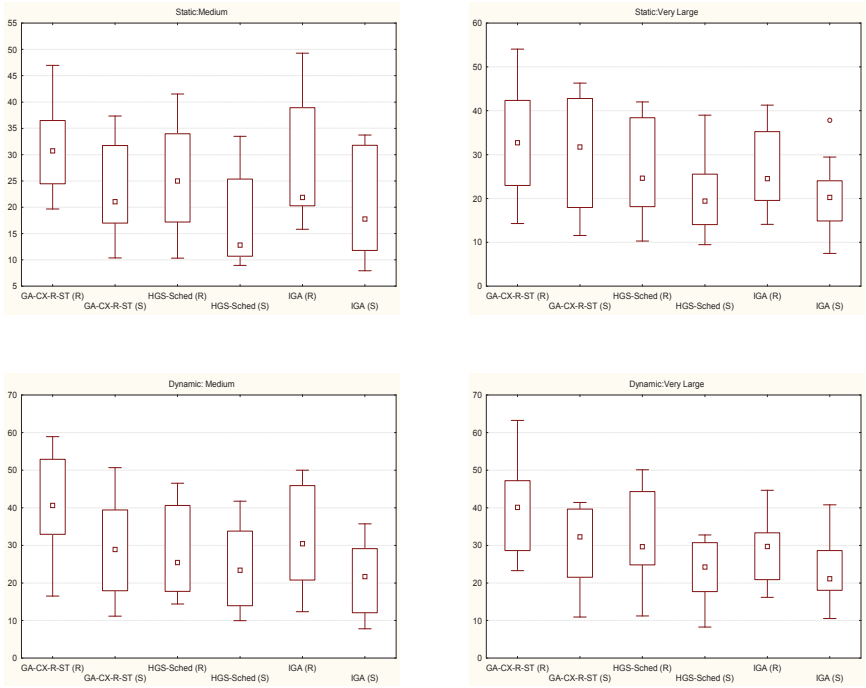

Fig. 4.7: The box-plot of the results for $Fail_r$ parameter in static and dynamic scenarios

with conventional scheduling objective functions such as *Makespan* and resource utilization.

One of the promising security-aware approaches in grid scheduling are based on the game-theoretical models. In [46] and [47] 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 [33] for simulating the resource owners selfish behavior. The results presented in [47] are extended by Wu et al. in [49]. The authors consider the heterogeneity of fault-tolerance mechanism in a security-assured Grid job scheduling and define four types of GA-based online schedulers for the simulation of fault-tolerance mechanisms. The other game-theoretical approaches are presented in [31] and [30]. In these papers the scheduling problem has been interpreted as a difficult decision problem for grid users working at different levels of the system. Users decisions and fundamental features arising in the users' behavior, such as cooperativeness, trustfulness and symmetric and asymmetric roles, were modelled based on the game theory paradigm. Another game-based model for security-aware grid scheduling is defined in [16]. The authors developed a node mobility prediction model for mobile grid systems and used the predetermined fair pricing strategies for secure access the mobile grid nodes. In all of the aforementioned models the final decisions on the

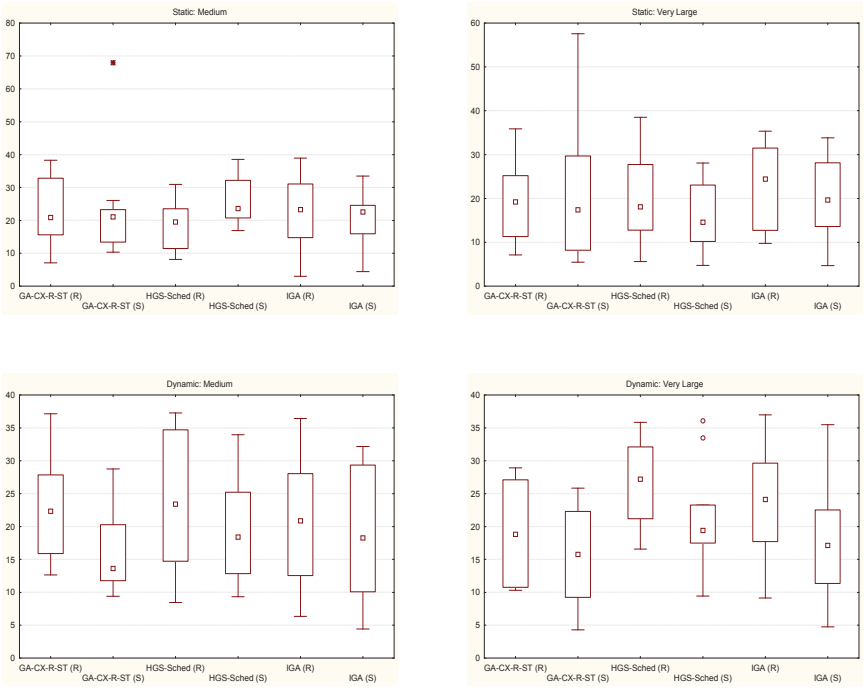

Fig. 4.8: The box-plot of the results for *Energy Improvement Rate* in static and dynamic scenarios

secure allocation of task to resources are made by the grid users who do not cooperate with each other. The costs of the risk-resilient tasks executions are interpreted as the users' cost functions, which are specified as the scheduling objectives and are minimized during the game. The main drawback of these approaches is their computational complexity. In many cases the games are provided on the different grid levels and to define an effective synchronization mechanism is a challenging task.

A significant volume of research has been done in the domain of energy aware resource management in modern large-scale distributed computational systems. Khan and Ahmad [25] have used the game theoretical methodologies to simultaneously optimize system performance and energy consumption in large data centers. Several research works have used similar models and approaches related to green computing in large-scale distributed systems, such as energy proportionality [19, 24], memory-aware computations, data intensive computations [26, 43], energy-efficient, and grid scheduling [35, 53]. (The recent survey is presented in [8].)

There is still not so large family of energy-aware genetic-based schedulers in grid and cloud environments. In most of the DVFS approached the scheduling has been defined as classical or dynamic load balancing problem. In such cases linear, dynamic and goal programming are the major optimization techniques (see i.e. [35], [24], [23], [26]).

Shen et al. in [44] and [45] present a *shadow price* technique for solving the scheduling problems in cloud systems. The "shadow price" for a pair task-machine is defined as an average energy consumption per instruction for the processor that can operate at different voltage levels. Then the classical move and swap mutation operations are used for an optimal mapping of tasks to machines.

Kessaci et al. in [22] present two versions of multi-objective parallel Genetic Algorithm (MOPGA) hybridized with energy-conscious scheduling heuristics (ECS). The GA engine is based on the concepts of island GA and multi-start GA models. The voltage and frequencies of the processors are scaled up at 16 discrete levels and genes in GA chromosomes are defined by the task-processor labels and processor voltage. The objective function is composed of two criteria: privileged makespan and total energy consumption in the system.

The solution presented in [22] is dedicated to general computing and embedded systems. An application of such methodology in computational cloud is demonstrated by Mezmaz et al. in [38]. The energy conservation rate in cloud system is very similar to the results obtained in the general case.

Another hybrid GA approach is presented by Miao et al. in [39]. The authors propose a multi-objective genetic algorithm which is hybridized with simulated annealing for the improvement of the local solutions.

4.9 Conclusions

The main reason behind the complexity of the multi-criteria grid scheduling is that this problems consist of several interconnected components (criteria, sub-problems), that can make many standard approaches ineffective. Even if the exact and efficient algorithms for solving particular components or aspects of an overall problem are well-known, these algorithms only yield solutions to sub-problems, and it remains an open question how to integrate these partial solutions to achieve a global optimum. Meta-heuristics, due to their robustness and high scalability, are able to tackle the various and also sometimes conflicting scheduling attributes and criteria.

The comprehensive empirical analysis presented in this chapter shows the high effectiveness of the single- and multi-population genetic-based metaheuristics in optimization of the conventional grid scheduling objectives, namely *Makespan* and *Flowtime*, as well as the new criteria such as the cumulative energy consumed in the grid system and the security issues. These additional criteria are usually considered as the separate optimization problems. Here in our approach they are embedded in the proposed scheduling models. The simulated grid scenarios in such cases can better illustrate the realistic systems, in which large number of variables, numerous objectives, constraints, and business rules, all contributing in various ways must be analyzed.

All models presented in this chapter are in fact not restricted just to the grid systems. They may be easily adapted to cloud environments, where security awareness and intelligent power management are the hottest research issues.

References

1. Abawajy, J. (2009): "An efficient adaptive scheduling policy for high performance computing", *Future Generation Computer Systems*, *Vol. 25, No. 3*, pp. 364–370.
2. Abraham, A., R. Buyya, and B. Nath (2000): "Nature's heuristics for scheduling jobs on computational grids", In *Proc. of the 8th IEEE International Conference on Advanced Computing and Communications, India*, pp. 45–52.
3. Aguirre, H. and K. Tanaka (2007): "Working principles, behavior, and performance of moeas on mnk-landscapes", *European Journal of Operational Research*, *Vol. 181*, pp. 1670–1690.
4. Ali, S., H. Siegel, M. Maheswaran, S. Ali, and D. Hensgen (2000): "Task execution time modeling for heterogeneous computing systems", In *Proc. of the Workshop on Heterogeneous Computing*, pp. 185–199.
5. Back, T., Fogel, D.B. and Michalewicz, Z. (Eds.): *Evolutionary Computation, Part 1 and 2* , IOP Publishing Ltd, 2000.
6. Baker, R. J. (2008): *CMOS: circuit design, layout, and simulation (Second ed.)*, Wiley.
7. Bartschi Wall, M. (1996): *A Genetic Algorithm for Resource-Constrained Scheduling*, PhD Thesis, Massachusetts Institute of Technology, MA.
8. Beloglazov, A., R. Buyya, Y. Lee, and A. Y. Zomaya (2011): "A taxonomy and survey of energy-efficient data centers and cloud computing systems", *Advances in Computers*, *Vol. 82*, pp. 47–111.
9. Brucker, P. (2007): *Scheduling Algorithms*, Springer Vlg.
10. Buyya, R. and M. Murshed (2002): "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing", *Concurrency and Computation: Practice and Experience*, *Vol. 14, No. 13–15*, pp. 1175–1220.
11. Carretero, J. and Xhafa, F. (2006): "Using Genetic Algorithms for Scheduling Jobs in Large Scale Grid Applications", *Journal of Technological and Economic Development –A Research Journal of Vilnius Gediminas Technical University*, Vol. 12, No. 1, pp. 11–17.
12. Davis, L. (Ed.) (1991): *Handbook of Genetic Algoriithms*, New York: Van Nostrand Reinhold.
13. Fibich, P. and Matyska, L. and Rudová, H. (2005): "Model of grid scheduling problem", in *Proc. of the AAAI-05 Workshop on Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing*.
14. Garg, S., R. Buyya, and H. Segel (2009): "Scheduling parallel aplications on utility grids: Time and cost trade-off management", in *Proc. of the 32nd ACSC, Wellington, Australia*, CRPIT, Vol. 91, Bernard Mans Ed.
15. Goldberg, D.E. and Lingle, R.Jr. (1985): "Alleles, loci and the travelling salesman problem", in *Proc. of the ICA 1985, Pittsburgh*.
16. Gosh, P. and S. Das (2010): "Mobility-aware cost-efficient job scheduling for single-class grid jobs in a generic mobile grid architecture", *Future Generation Computer Systems*, *Vol. 26, No. 8*, pp. 1356–1367.
17. Graham, R., E. Lawler, J. Lenstra, and A. Rinnooy Kan (1979): "Optimization and approximation in deterministic sequencing and scheduling: a survey", *Annals of Discrete Mathematics*, *Vol. 5*, pp. 287–326.
18. Grueninger, T. (1997): *Multimodal optimization using genetic algorithms*, Technical report, Department of Mechanical Engineering, MIT, Cambridge, MA.
19. Guzek, K., J. E. Pecero, B. Dorrosoro, P. Bouvry, and S. U. Khan (2010): "A cellular genetic algorithm for scheduling applications and energy-aware communication optimization", In *Proc. of the ACM/IEEE/IFIP Int. Conf. on High Performance Computing and Simulation (HPCS), Caen, France*, pp. 241–248.
20. Humphrey, M. and M. Thompson (2001): "Security implications of typical grid computing usage scenarios", In *Proc. of the Conf. on High Performance Distributed Computing*.
21. Hwang, S. and C. Kesselman (2003): "A flexible framework for fault tolerance in the grid", *J. of Grid Computing*, *Vol. 1, No. 3*, pp. 251–272.
22. Kessaci, Y., M. Mezmaz, N. Melab, E.-G. Talbi, and D. Tuyttens (2011): "Parallel evolutionary algorithms for energy aware scheduling", In *Intelligent Decisions Systems in Large-Scale*

Distributed Environments, P.Bouvry, H. Gonzalez-Velez and J. Kołodziej (Eds), Studies in Computational Intelligence, Springer Vlg, Volume Vol. 362, Chapter 4.
23. Khan, S. (2009a): "A goal programming approach for the joint optimization of energy consumption and response time in computational grids", In *Proc. of the 28th IEEE International Performance Computing and Communications Conference (IPCCC), Phoenix, AZ, USA*, pp. 410–417.
24. Khan, S. (2009b): "A self-adaptive weighted sum technique for the joint optimization of performance and power consumption in data centers", In *Proc. of the 22nd International Conference on Parallel and Distributed Computing and Communication Systems (PDCCS), USA*, pp. 13–18.
25. Khan, S. U. and I. Ahmad (2009): "A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids", *IEEE Tran.on Parallel and Distributed Systems*, *Vol. 20, No. 3*, pp. 346–360.
26. Kliazovich, D., P. Bouvry, and S. U. Khan (2010): "Dens: Data center energy-efficient network-aware scheduling", In *Proc. of ACM/IEEE International Conference on Green Computing and Communications (GreenCom), Hangzhou, China, December 2010*, pp. 69–75.
27. Klusaček, D. and H. Rudová (2011): "Efficient grid scheduling through the incremental schedule-based approach", *Computational Intelligence*, *Vol. 27, No. 1*, pp. 4–22.
28. Kołodziej J. (2012): *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid System*, *Studies in Computational Intelligence*, Vol. 419, Springer Vlg.
29. 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 (2011)*, pp. 1035–1046.
30. Koł odziej, J. and F. Xhafa (2011). "Integration of task abortion and security requirements in ga-based meta-heuristics for independent batch grid scheduling", *Computers and Mathematics with Applications*, *Vol. 63 (2012)*, pp. 350–364.
31. Kołodziej, J. and F. Xhafa (2011): "Meeting security and user behaviour requirements in grid scheduling", *Simulation Modelling Practice and Theory*, *Vol. 19, No. 1*, pp. 213–226.
32. Kołodziej, J. and F. Xhafa (2011): "Modern approaches to modelling user requirements on resource and task allocation in hierarchical computational grids", *Int. J. on Appled Mathematics and Computer Science*, *Vol. 21, No. 2*, pp. 243–257.
33. Kwok, Y.-K., K. Hwang, and S. Song (2007): "Selfish grids: Game-theoretic modeling and nas/psa benchmark evaluation", *IEEE Tran. on Parallel and Distributing Systems*, *Vol. 18, No. 5*, pp. 1–16.
34. Lapin, L. (1998): *Probability and Statistics for Modern Engineering (2nd Ed.)*.
35. Lee, Y. C. and A. Y. Zomaya (2009): "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling", In *Proc. of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid CCGrid), Shanghai, China*, pp. 92–99.
36. Mann, P. S. (2010). *Introductory Statistics, 7th ed.* Wiley.
37. Mejia-Alvarez, P., E. Levner, and D. Mossé (2004): "Adaptive scheduling server for power-aware real-time tasks", *ACM Trans. Embed. Comput. Syst.*, *Vol. 3, No. 2*, pp. 284–306.
38. Mezmaz, M., N. Melab, Y. Kessaci, Y. Lee, E.-G. Talbi, A. Zomaya, and D. Tuyttens (2011): "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems", *J. Parallel Distrib. Comput.*, *doi:10.1016/j.jpdc.2011.04.007*.
39. Miao, L., Y. Qi, D. Hou, Y. Dai, and Y. Shi (2008): "A multi-objective hybrid genetic algorithm for energy saving task scheduling in cmp system", In *Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics (ICSMC2008)*, pp. 197–201.
40. Michalewicz, Z. (1992): *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Vlg.
41. Muhammad, R., F. Hussain, and O. Hussain (2011): "Neural network-based approach for predicting trust values based on non-uniform input in mobile applications", *The Computer Journal online*.
42. Olivier, I, Smith, D. and Holland, J. (1987): "A study of permutation crossover operators on the travelling salesman problem", in *Proc. of the ICGA 1987, Cambridge, MA*, pp. 224–230.

43. Pinel, F., J. Pecero, P. Bouvry, and S. U. Khan (2010): "Memory-aware green scheduling on multi-core processors", In *Proc. of the 39th IEEE International Conference on Parallel Processing (ICPP)*, pp. 485–488.
44. Shen, G. and Y. Zhang (2011): "A new evolutionary algorithm using shadow price guided operators", *Applied Soft Computing*, *Vol. 11, No. 2*, pp. 1983–1992.
45. Shen, G. and Y. Zhang (2011): "A shadow price guided genetic algorithm for energy aware task scheduling on cloud computers", In *Proc. of the 3rd International Conference on Swarm Intelligence - (ICSI-2011)*, pp. 522–529.
46. Song, S., K. Hwang, and Y. Kwok (2005): "Trusted grid computing with security binding and trust integration", *J. of Grid Computing*, *Vol. 3, No. 1-2*, pp. 53–73.
47. Song, S., K. Hwang, and Y. Kwok (2006): "Risk-resilient heuristics and genetic algorithms for security- assured grid job scheduling", *IEEE Tran. on Computers*, *Vol. 55, No. 6*, pp. 703—719.
48. Whitley, D., Rana, S. and R. Heckendorn (1998). "The island model genetic algorithm: On separability, population size and convergence", *Journal of Computing and Information Technology*, *Vol. 7*, pp. 33–47.
49. Wu, C.-C. and R.-Y. Sun (2010): "An integrated security-aware job scheduling strategy for large-scale computational grids", *Future Generation Computer Systems*, *Vol. 26, No. 2*, pp. 198–206.
50. Xhafa, F. and A. Abraham (2010): "Computational models and heuristic methods for grid scheduling problems", *Future Generation Computer Systems*, *Vol. 26 (2010)*, pp. 608–621.
51. Xhafa, F., J. Carretero, and A. Abraham (2007): "Genetic algorithm based schedulers for grid computing systems", *Int. J. of Innovative Computing, Information and Control*, *Vol. 3, No. 5*, pp. 1053–1071.
52. Xhafa, F., J. Carretero, L. Barolli, and A. Durresi (2007): "Requirements for an event-based simulation package for grid systems", *Journal of Interconnection Networks*, *Vol. 8, No. 2*, pp. 163–178.
53. Zomaya, A., Y. (2009): "Energy-aware scheduling and resource allocation for large-scale distributed systems", In *11th IEEE International Conference on High Performance Computing and Communications (HPCC), Seoul, Korea*.