Cluster Comput (2013) 16:591–609
DOI 10.1007/s10586-012-0226-7

Hierarchical genetic-based grid scheduling with energy optimization

Joanna Kołodziej · Samee Ullah Khan · Lizhe Wang · Aleksander Byrski · Nasro Min-Allah · Sajjad Ahmad Madani

Received: 20 January 2012 / Accepted: 30 June 2012 / Published online: 11 August 2012

Abstract An optimization of power and energy consumptions is the important concern for a design of modern-day and future computing and communication systems. Various techniques and high performance technologies have been investigated and developed for an efficient management of such systems. All these technologies should be able to provide good performance and to cope under an increased workload demand in the dynamic environments such as Computational Grids (CGs), clusters and clouds.

In this paper we approach the independent batch scheduling in CG as a bi-objective minimization problem with makespan and energy consumption as the scheduling criteria. We use the *Dynamic Voltage Scaling (DVS)* methodology for scaling and possible reduction of cumulative power energy utilized by the system resources. We develop two implementations of Hierarchical Genetic Strategy-based grid scheduler (*Green-HGS-Sched*) with elitist and struggle replacement mechanisms. The proposed algorithms were empirically evaluated versus single-population Genetic Algorithms (GAs) and Island GA models for four CG size scenarios in static and dynamic modes. The simulation results show that proposed scheduling methodologies fairly reduce the energy usage and can be easily adapted to the dynamically changing grid states and various scheduling scenarios.

Keywords Genetic algorithm · Hierarchical genetic strategy · Computational grid · Scheduling · Dynamic voltage · Frequency scaling

J. Kołodziej (✉)
Institute of Computer Science, Cracow University of Technology, ul. Warszawska 24, 31-155 Cracow, Poland
e-mail: jkolodziej@uck.pw.edu.pl

S.U. Khan
NDSU-CIIT Green Computing and Communications Laboratory, North Dakota State University, Fargo, ND 58108, USA
e-mail: samee.khan@ndsu.edu

L. Wang
Center for Earth Observation, Chinese Academy of Sciences, Beijing, China
e-mail: LZWang@ceode.ac.cn

A. Byrski
AGH University of Science and Technology, Cracow, Poland
e-mail: olekb@agh.edu.pl

N. Min-Allah · S.A. Madani
Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, Abbottabad, Pakistan

N. Min-Allah
e-mail: nasar@comsats.edu.pk

S.A. Madani
e-mail: madani@ciit.net.pk

1 Introduction

Grid computing has emerged as a wide area distributed platform for solving the large-scale problems in science, engineering, etc. Computational Grid (CG) involves the combination of many computing resources into a network for the execution of computational tasks. The resources are distributed across multiple organizations, administrative domains having their own access, usage policies and local schedulers. The tasks scheduling and the effective management of the resources in such systems remain complex problems and therefore, demands sophisticated tools for analyzing the algorithms performances before applying them to the real systems.

The main issues related to energy efficiency have been introduced by the large scales of enterprise computing environments and data centers. Due to the importance of power

and energy consumption in modern-day and future computing and communication systems various techniques and recent technologies have been investigated and developed. However, these solutions are mainly related to an optimization of the system thermodynamics [37]. It requires profiles of hardware energy consumption and application energy consumption, and the correlation between workload distribution and the energy consumption of power and cooling [12].

While the CGs have been widely promoted as cheap alternative to supercomputers, a significant disproportion of resource availability and resource provisioning may be observed in the system [21]. Therefore, the current efforts in the grid computing research focus on the design of new effective grid schedulers, that can simultaneously optimize the key grid objectives, such as makespan, flowtime, resource and cumulative energy utilization [11]. Energy efficient scheduling in CGs becomes a complex endeavor due to the multi-constraints, various optimization criteria and different priorities of the resource owners. Various types of information and data processed in the large-scale dynamic grid environment may be incomplete, imprecise, fragmentary and overloading, which complicates the assignment scores, availability of resources, and the may increase the amount of energy used in the system [43]. Heuristic approaches have shown great potential to solve many demanding, real-world decision and optimization problems in uncertain large-scale environments and seem to be the effective means for designing energy-aware grid schedulers in CGs [27, 28].

The main objective of this work is to define an effective genetic-based batch scheduler, that can be easily implemented in the dynamic grid environment and enable an energy aware allocation of the grid resources. We address a *Independent Batch Scheduling* problem in CGs, where tasks are processed in a batch mode and there are no dependencies among them. This scheduling scenario is very useful in illustrating many realistic grid approaches[ref]. We define two main scheduling criteria, which are optimized in hierarchical mode, namely *makespan* as the privileged criterion and *average energy consumption*. We use a *Dynamic Voltage Scaling (DVS)* methodology for reducing the cumulative power energy utilized by the system resources. Based on the result of our preliminary study on the effectiveness of mono-population genetic-based schedulers in energy-aware scheduling in grids [26, 29], we developed two implementations of hierarchical *Green-HGS-Sched* genetic scheduler and provided the empirical evaluation in two "energetic" scheduling modes in static and dynamic grid scenarios. The performance of these hierarchical schedulers have been measured by using the makespan and relative energy consumption improvement rate metrics. The effectiveness of the implementations of *Green-HGS-Sched* were compared with the results achieved by four single-population Genetic Algorithms (GAs) and Island GA scheduler [46]. All schedulers have been integrated with the grid simulator.

The remainder of this paper is structured as follows. Related work is discussed in Sect. 2 and the addressed scheduling problem is specified in Sect. 3. The generic energy model is defined in Sect. 5 together with the main scheduling scenarios and criteria. The *Green-HGS-Sched* framework and genetic operators are presented in Sect. 7. Section 8 presents the results of a simple empirical analysis of the effectiveness of hierarchical, island and mono-population schedulers. The paper is summarized in Sect. 9.

2 Related work

Numerous interesting research projects have been recently realized in the area of energy aware resource management in modern large-scale distributed computing system. Based on the taxonomy defined for cloud computing in [15], the power and energy management methodologies in distributed computing environments can be classified into two main categories, namely *static energy management (SEM)* methods and *dynamic energy management (DEM)* techniques, as it is presented in Fig. 1 (see also [28]).

The static management methodologies are working usually at the hardware level of the class of the static management In such systems, the physical computational devices can be replaced by the low-power battery machines or nanoprocessors and the system workload can be effectively distributed. It allows to optimize the energy utilized for computing the applications, storage and data transfer by reducing the number of idle devices and idle periods of active processors. The major projects based on the static power management include Green-Destiny [45], FAWN [3] or Gordon [10] projects.

Dynamic Voltage and Frequency Scaling method became recently a key dynamic power management methodology supporting the energy efficient scheduling in grids and large-scale data systems [30, 49]. In most of the DVFS approached the scheduling has been defined as classical or dynamic load balancing problem. Khan and Ahmad [21] have successfully used the game theory paradigm for the optimization of the system performance and energy consumption. Several research works have used similar models and approaches, that have addressed various research problems related to large-scale computing systems, such as energy proportionality [17, 20], memory-aware computations, data intensive computations, energy-efficient, and grid scheduling [22, 39]. A lot of interesting examples of recently developed static and dynamic power and energy management techniques in the distributed computing environments are presented in the following surveys [5, 40, 43, 44].

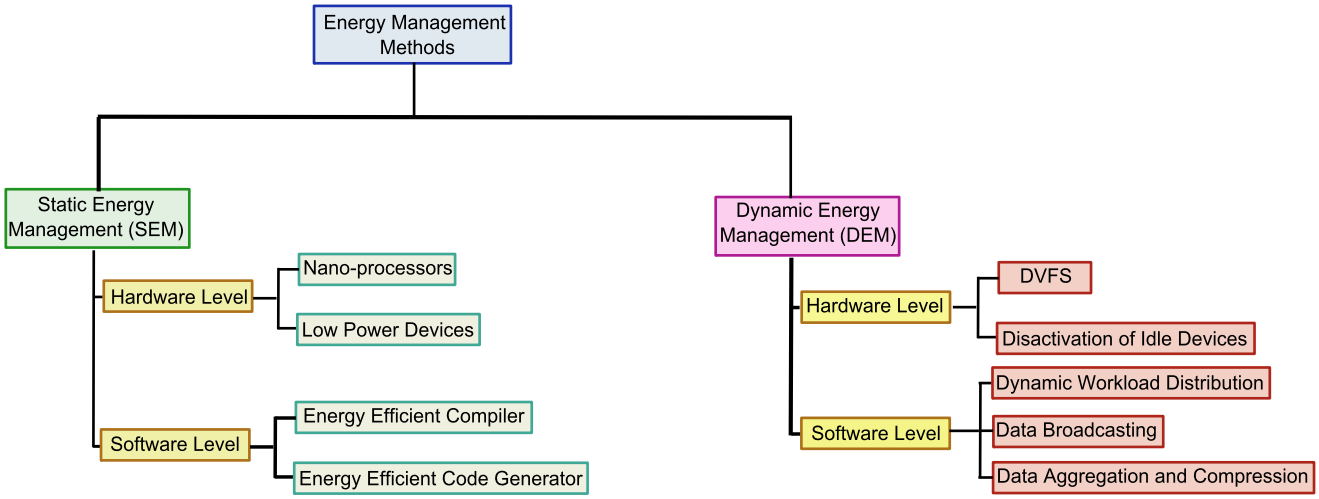


Fig. 1 Taxonomy of energy and power management techniques in large-scale distributed computing environments [28]

Although a significant volume of the research has been provided in energy effective scheduling and resource allocation in large-scale computing systems, still not so large family of energy-aware genetic-based grid and cloud schedulers have been developed. Most of those approaches need an implementation of specially designed genetic operators, such as partially matching or cycle crossover and swap or rebalancing mutation mechanisms primarily designed for solving the complex combinatorial optimization problems [25]. An energy consumed by the system is usually just one of the components of a multi-objective fitness function.

In [41] and [42] Shen et al. present a *shadow price* technique for improving the genetic operations in standard GA used as a scheduler in computational cloud. 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. The fitness function for such GA scheduler is expressed as a total energy consumption.

Kessaci et al. in [19] 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 authors consider parallel applications represented by a directed acyclic graph (DAG), which are mapped onto multi-processors machines. 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 reduction of the energy utilization achieved in the experimental analysis is about 47.4 %.

The solution presented in [19] is dedicated to general computing and embedded systems. An application of such methodology in computational cloud is demonstrated by Mezmaz et al. in [34]. 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 [35]. The authors propose a multi-objective genetic algorithm which is hybridized with simulated annealing for the improvement of the local solutions.

3 Independent batch scheduling problem in computational grids

Due to the high parametrization, sheer size and dynamics of the grid system, scheduling problems in grids may be considered in fact as a family of NP-complete optimization problems [13]. Depending on the requirements of the grid users, the complexity of the problem can be determined by the number of objectives to be optimized, the type of the environment (static or dynamic), task processing (immediate or batch), task interrelations (independence or dependency), grid resource management (centralized, decentralized and hierarchical), and many others. To achieve the desired performance of the system, both users' conditions and grid environment information must be "embedded" into the scheduling mechanism [1], [25].

The main attributes of the gird scheduling are presented in Fig. 2.

In this paper we address an *Independent Batch Scheduling* problem in CGs. In this problem, it is assumed that tasks are grouped into batches and can be executed independently in static or dynamic grid environments. The scheduling attributes needed for the specification of this problem

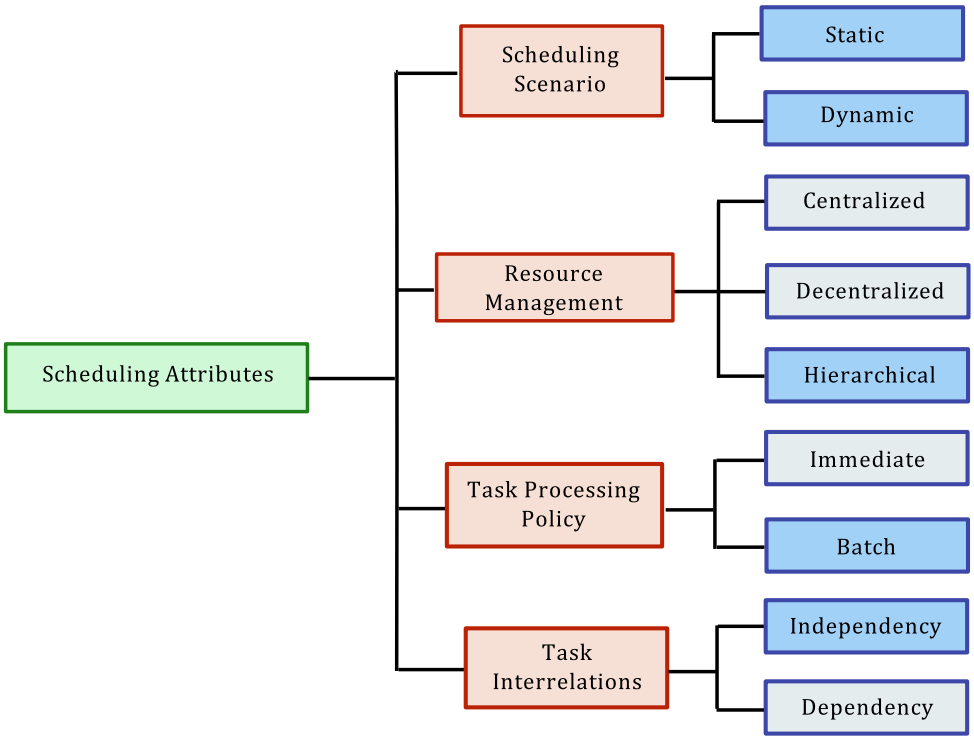


Fig. 2 Main scheduling attributes in CGs

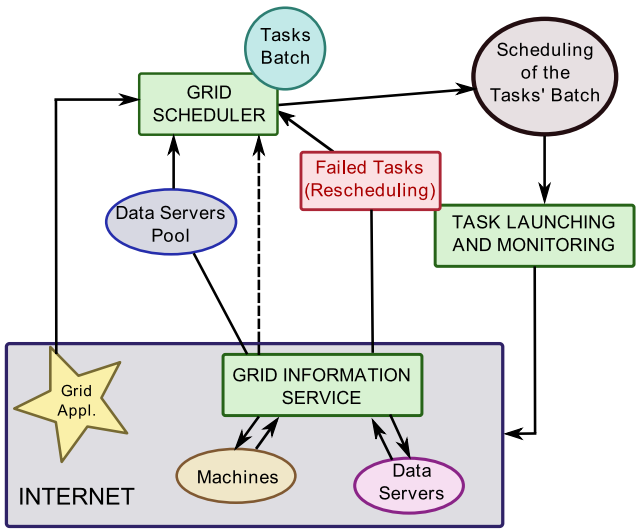


Fig. 3 Main phases of the batch scheduler in CGs

are highlighted in Fig. 2 as the dark blue text boxes. The generic independent batch scheduling model is effective in massive parallel processing of the applications that require a large amount of data. Therefore, there are many realistic scenarios, including banking systems, virtual campuses, health systems, bio-informatics applications, and many others, where independent batch scheduling is successfully applied. However, even under the independent nature of tasks and the batch processing, the problem is computationally hard to solve.

The independent batch scheduling procedure can be realized in the following six steps:

(i) Get the information on available resources;
(ii) Get the information on pending tasks;
(iii) Get the information on data hosts where data files for tasks completion are required;
(iv) Prepare a batch of tasks and compute a schedule for that batch on available machines and data hosts;
(v) Allocate tasks;
(vi) Monitor (failed tasks are re-scheduled).

In Fig. 3 we present a simple graphical flowchart of the batch scheduling phases.

To our best knowledge there is no standard notation for classification of the scheduling problems in CGs. A simple extension of conventional Graham's [16] and Brucker's [8] classifications of scheduling problems has been proposed by Fibich et al. [14]. The characteristics of the resource-constrained project scheduling problem [9] and resource-constrained machine scheduling [6, 7] may be helpful in the specification and formal description of the grid resources.

Based on the methodology presented in [14] and [23], and the main scheduling attributes presented in Fig. 2, the instance of the independent batch grid scheduling problem considered in this paper can be denoted by using the following expression:

$$Rm\big[\{b, indep, (stat, dyn), h\}\big]\big(C_{max}, E_I(E_{II})\big) \tag{1}$$

where:

- Rm—according to the Graham's notation, it means that the tasks are mapped into the (parallel) resources of various speed[1];
- b—means that the task processing mode is "batch mode";
- $indep$—denotes "independency" as the task interrelation;
- $(stat, dyn)$—means that we will consider both static and dynamics grid scheduling modes;
- h—means that the scheduling objectives are optimized in hierarchical mode;
- $C_{\max}$—denotes a makespan as the privileged scheduling objective;
- $E_I(E_{\text{batch}})$—denotes total energy consumption as the second scheduling criterion (E_I or E_{batch} is selected depending on the scheduling scenario (see Sect. 6.4)).

Most of these parameters will be explained in Sect. 4, 5 and 6.

4 Expected time to compute (ETC) matrix model

In order to estimate the execution times of tasks on machines we used the *Expected Time to Compute (ETC)* matrix model [2] adopted to the independent batch scheduling. It is assumed in this model that each task can only be executed on one grid node in each batch and no preemptive process is allowed within tasks or resources. In the case of the failures of machines, the tasks are re-scheduled in the next batch, however, the scheduling of tasks in different batches are the independent processes. It is also assumed that when a machine processes its tasks, there is no priority distinctions between the tasks assigned in the previous batches and those assigned in the current batch. And finally, each machine cannot remain idle and all tasks assigned to this machine must be activated.

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

The following notation for tasks and machines will be used throughout the paper [26, 27]:

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

The tasks and machines in the grid systems are characterized by the following general parameters:

(a) *Task* j:

- wl_j—load parameter expressed in Millions of Instructions (MI)– we denote by $WL = [wl_1, \ldots, wl_n]$ 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), we denote by $CC = [cc_1, \ldots, cc_m]$ a *computing capacity vector*;
- $ready_i$—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 $ready\_times = [ready_1, \ldots, ready_m]$.

In this generic model there is no detailed specification of the types of tasks and machines. The tasks can be considered as monolithic applications or large-scale metatasks with no dependencies among the components. The workloads of tasks can be estimated based on the specifications provided by the users, on historical data, or it can be generated based on the system predictions [18]. As machines we usually define the multiprocessors or parallel machines (see Rm parameter in the notation) or even small local area networks or computational clusters.

For each task-machine pair, the coordinates of WL and CC vectors can be used for an estimation of the completion times of the task j on machine i. These completion times are denoted by $ETC[j][i]$ ($i \in M_l$, $j \in N_l$), and can be calculated in the following way:

$$ETC[j][i] = \frac{wl_j}{cc_i}. \tag{2}$$

All $ETC[j][i]$ parameters are defined as the elements of an ETC matrix, $ETC = [ETC[j][i]]_{n\times m}$, which is the main structure in ETC model.

In simulation analysis, wl_j and cc_i parameters are usually generated by using the Gamma probability distribution (or the standard Gauss distribution) [32] in order to express the heterogeneities of tasks and machines in the grid system (see Sect. 8.1).

5 Energy model

The energy model presented in this paper is based on the power consumption model in complementary metal-oxide semiconductor (CMOS) logic circuits [4]. In this model, the capacitive power P_{ij} 'consumed by the machine i for computing the task j is calculated in the following way:

$$P_{ij} = A \cdot C \cdot v^2 \cdot f, \tag{3}$$

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 machine's frequency. It is assumed that operating frequency of each machine is approximately proportionate to its processing speed (see [33]). The decrease in the supply voltage and frequency reduces the energy consumed by the machine.

We assume that each machine in the grid system is equipped with *Dynamic Voltage and Frequency Scaling (DVFS)* module [31], that allows the modulation of the supply voltage and operating frequency of this machine. In Table 1 we present the parameters for 16 DVFS levels, that specify three "energetic" categories for grid machines in our system (see also [34]).

For each machine i ($i = 1, \ldots, m$), its "energetic" class is denoted by s^i and is characterized by the following column meta-vector $Vr_{(i)}$ of DVFS levels with different (reduced) values of the supply voltage and the corresponded relative machine frequency parameters:

$$Vr_{(i)} = \left[(v_{s_0}(i), f_{s_0}(i)); \ldots; (v_{s_{l(\max)}}(i), f_{s_{l(\max)}}(i))\right]^T \tag{4}$$

For lower supply voltage, the operation frequency of the machine decreases, which means that $f_{s_l}(i)$ coefficients are within the range of [0, 1]. We assume in this work that the supply voltage is constant during the calculation (execution) of each task, but may be different for different tasks.

The decreasing of the machine frequency and supply voltage leads to the increased computational times of the tasks executed on the machine. For a given task-machine pair (j, i), the time of completion the task j on machine i at various DVFS levels specified for the class s^i) can be defined as the coordinates of a vector $\widehat{ETC}[j][i]$ and calculated by using the following formula:

$$\widehat{ETC}[j][i] = \left[\frac{1}{f_{s_0}(i)} \cdot ETC[j][i], \ldots, \frac{1}{f_{s_{l(\max)}}(i)} \cdot ETC[j][i]\right] \tag{5}$$

where $l(\max)$ denotes a number of DVFS levels in the class s^i, $ETC[j][i]$ is the task execution time calculated according to the (2) and $\{f_{s_0}(i), \ldots, f_{s_{l(\max)}}(i)\}$ are the relative frequencies of the machine i, specified for the class s^i at the $s_0, \ldots, s_{l(\max)}$ DVFS levels.

It can be observed from the (5) that the inversions of the relative frequency coefficients approximately estimates the raises of the completion times of tasks on machines. It is a consequence of the previous assumption about the inverse proportion of the completion times of tasks and the frequencies of machines.

The ETC matrix can be easily adapted to the energy-aware scheduling model. In such a case an ETC meta-matrix is defined based on the standard ETC matrix, where each $ETC[j][i]$ element is replaced by the corresponding $\widehat{ETC}[j][i]$ vector (for each pair (j, i)), that is to say:

$$\widehat{ETC} = \left[\widehat{ETC}[j][i][s_l]\right]_{n \times m \times s_{l(\max)}} \tag{6}$$

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

Based on (6) and (3) we can express the energy consumed for completing the task j on machine i at the level s_l, as a scalar product of the number of switches per clock cycle, total capacitance load, frequency and squared voltage at a given level s_l and the estimated completion time, that is to say:

$$E_{ji} = \gamma \cdot (f_{s_l}(i))_j \cdot f \cdot \left[(v_{s_l}(i))_j\right]^2 \cdot \widehat{ETC}[j][i][s_l] \tag{7}$$

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

Based on (6), (7) and (5) the computational times for each possible pair (j, i) at the level s_l can be calculated as follows:

$$\begin{aligned} E_{\{j,i,l\}} &= \gamma \cdot \left(f_{s_l}(i)\right)_j \cdot f \cdot \left[\left(v_{s_l}(i)\right)_j\right]^2 \cdot \left(f_{s_1}(i)\right)_j \cdot ETC[j][i] \\ &= \gamma \cdot f \cdot \left[\left(v_{s_l}(i)\right)_j\right]^2 \cdot ETC[j][i] \end{aligned} \tag{8}$$

The cumulative energy consumed 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:

$$\begin{aligned} E_i = &\sum_{\substack{j \in T(i) \\ l \in L_j}} \{E_{\{j,i,l\}}\} + \gamma \cdot f \cdot [v_{s_{\max}}]^2 \cdot ready_i \\ &+ \gamma \cdot f_{s_{\min}}(i) \cdot f\left[v_{s_{\min}}(i)\right]^2 \cdot Idle[i] \\ = &\gamma \cdot f \sum_{\substack{j \in T(i) \\ l \in L_i}} \left(\left[\left(v_{s_l}(i)\right)_j\right]^2 \cdot ETC[j][i]\right) \\ &+ \left[v_{s_{\max}}(i)\right]^2 \cdot ready_i + f_{s_{\min}}(i) \cdot \left[v_{s_{\min}}(i)\right]^2 \cdot Idle[i] \end{aligned} \tag{9}$$

where $T(i)$ is a set of tasks assigned to the machine i, $ready_i$ is the ready time of the machine i, $Idle[i]$ denotes an idle time of the machine i and L_i denotes a subset of DVFS levels used for the tasks assigned to machine i. We ignore all additional machine frequency transition overheads, which take usually a negligible amount of time (e.g., 10 ms–150 ms, see [38]) and do not bear down on the overall ETC model with an active 'energetic' module.

Table 1 DVFS levels for three machine classes

Level	Class I		Class II		Class III	
	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				

And finally, we can complete our formal description of the DVFS-based energy model adapted to the independent batch scheduling in grids with a definition of an average cumulative energy utilized by the grid system for completion of all tasks in the batch:

$$E_{\text{batch}} = \frac{\sum_{i=1}^{m} E_i}{m} \tag{10}$$

6 Scheduling scenarios and objectives

The DVFS-based energy model for grid system defined in the previous section, will be used now for the specification of two scheduling scenarios and for the definition of the scheduling criteria.

6.1 Scheduling representation

The solutions of the scheduling problem addressed in this paper (schedules) can be encoded as the permutation strings (with and without repetitions) of task and machine labels. We consider in this paper two different encoding methods of schedules, namely *direct representation* and *permutation-based representation*.

In direct representation schedules are the elements of the set of all permutations of the length n, *with repetitions*, over the set of machine labels M_l. We denote this set by $\mathcal{S}$. Formally, each schedule $S \in \mathcal{S}$ it is encoded by the following vector:

$$S = [i_1, \ldots, i_n]^T \tag{11}$$

where $i_j \in M_l$ is a label of the machine on which the task j is computed.

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

The direct representation of the schedules can be easily transformed into a *permutation-based representation*, in which, for each machine, a sequence of tasks assigned to that machine is specified. The tasks in the sequence are sorted (in increasing order) with respect to their completion times. Thereafter, all of the task sequences are concatenated into a vector u, which is in fact a permutation without repetition of tasks to machines. Formally, in this case the codes of schedules are the elements of a set $\mathcal{S}_{(1)}$ of all permutations of the length n, *without repetitions*, over the set of task labels N_l, and are defined as the following vectors:

$$u = [u_1, \ldots, u_n]^T \tag{12}$$

where $u_i \in N_l$, $i = 1, \ldots, n$.

The cardinality of $\mathcal{S}_{(1)}$ is $n!$.

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

6.2 Scheduling scenarios

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

Two basic models are utilized in multi-objective optimization: hierarchical and simultaneous modes. In the *simultaneous mode* (s) all objective functions are optimized simultaneously while in the *hierarchical* (h) case, the objectives are sorted *a priori* according to their importance in the model. The process starts by optimizing the most important criterion. When further improvements are impossible, the second criterion is optimized under the restriction of keeping unchanged (or improving) the optimal values of the first one. In this paper we define the scheduling problem as a discrete 2-step hierarchical global optimization procedure, where: (i) a *makespan* is considered as a dominant criterion, and it is minimized in the first step, and (ii) in the second step a *total energy consumption* is minimized with the assumption that the makespan value does not increase.

One of the main objectives of our work is to compare the results of the scheduling in grids in two following scenarios:

1. I—*Max-Min Mode*, in which each machine works at the maximal DVFS level during the computations and turns into the idle mode after the execution of all tasks assigned to this machine;

2. II—*Modular Power Supply Mode*, in which each machine may work at different DVFS levels during the task executions and then may turn into the idle mode.

The first mode seems to be the most effective in the case of low-power devices or services defined as "machines" (resources) in the system. No modification of the standard scheduling procedures and standard scheduling objectives, such as makespan, flowtime, tardiness, etc., is necessary. The second mode may be a good candidate for a testbed architecture for the future generation grid systems. The optimal power supply levels can be specified for each current devices (machines), that can be in the future replaced by the next-generation low-power devices for keeping (or improving) the energy consumption at optimal level.

In the following two subsections we define the procedures of calculation of the makespan and total energy consumed in the system in these two above mentioned scenarios.

6.3 Makespan optimization

Makespan is expressed as a finishing time of the latest task in the batch. That is to say:

$$C_{\max} = \min_{S \in Schedules} \max_{j \in N} C_j \tag{13}$$

where C_j denotes the time needed for finalizing the task j.

Using the ETC matrix model, the makespan can be defined in terms of the completion times of the machines. The time of finishing the last task is specified as the maximal completion time of the machines available for the batch of tasks. Let us denote by $completion[i]$ *a completion time of machine* i, which is a cumulative time necessary for reloading the machine i after finalizing the previously assigned tasks and for completing the tasks actually assigned to the machine. In *Max-Min Mode* this completion time can be defined as follows:

$$completion_I[i] = ready_i + \sum_{j \in T(i)} ETC[j][i] \tag{14}$$

The makespan in this mode is defined in the following way:

$$(C_{\max})_I = \max_{i=1}^{m} completion_I[i] \tag{15}$$

The idle time for machine i working in *Max-Min Mode* can be calculated as a difference between the makespan and $completion_I[i]$, that is to say:

$$Idle_I[i] = (C_{\max})_I - completion_I[i] \tag{16}$$

For the machine with the maximal completion time (makespan) the idle factor is zero.

In order to define the makespan in *Modular Power Supply Mode*, we must specify the actual DSVF level s_l for a given machine. The formulas for computing the completion time, makespan and idle time can be defined in the following way:

$$completion_{II}[i] = ready_i + \sum_{j \in T(i)} \frac{1}{f_{s_l}}(i) \cdot ETC[j][i]. \tag{17}$$

$$(C_{\max})_{II} = \max_{i=1}^{m} completion_{II}[i] \tag{18}$$

$$Idle_{II}[i] = (C_{\max})_{II} - completion_{II}[i] \tag{19}$$

6.4 Energy optimization

The second step of the scheduling optimization procedure is the minimization of the total energy consumed in CG for scheduling a given batch of tasks. We assume the minimal power supply for each machine in the idle mode and maximal power and voltage supply in reloading process.

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

$$E_I = \frac{1}{m} \cdot \sum_{i=1}^{m} \gamma \cdot completion_I[i] \cdot f \cdot \left[v_{s_{\max}}(i)\right]^2 + \frac{1}{m} \cdot \sum_{i=1}^{m} \gamma \cdot f_{s_{\min}}(i) \cdot \left[v_{s_{\min}}(i)\right]^2 \cdot Idle_I[i] \tag{20}$$

In *Modular Power Supply Mode* the average cumulative energy is given by the (10), that is to say:

$$E_{II} = E_{\text{batch}} = \frac{\sum_{i=1}^{m} E_i}{m} \tag{21}$$

where[2]

$$E_i = \gamma \cdot f \cdot \sum_{\substack{j \in T(i) \\ l \in L_i}} \left(\left[\left(v_{s_l}(i)\right)_j\right]^2 \cdot ETC[j][i]\right) + \left[v_{s_{\max}}(i)\right]^2 \cdot ready_i + f_{s_{\min}}(i) \cdot \left[v_{s_{\min}}(i)\right]^2 \cdot Idle[i] \tag{22}$$

In both cases E_I and E_{II} are minimized subject to the following constraint:

$$\sum_{l \in L_i} \left[\frac{1}{f_{s_l}(i)} \cdot ETC[j][i]\right] \le C_{\max}; \quad \forall i \in \{1, \ldots, m\} \tag{23}$$

where L_i denotes a subset of DVFS levels specified for tasks assigned to the machine i.

7 *Green-HGS-Sched*: energy-aware hierarchical genetic scheduler

An exploration of the search space in grid scheduling is very complex, mainly because of the sheer size of the solution space and the system dynamics. The search space is

[2] See (9).

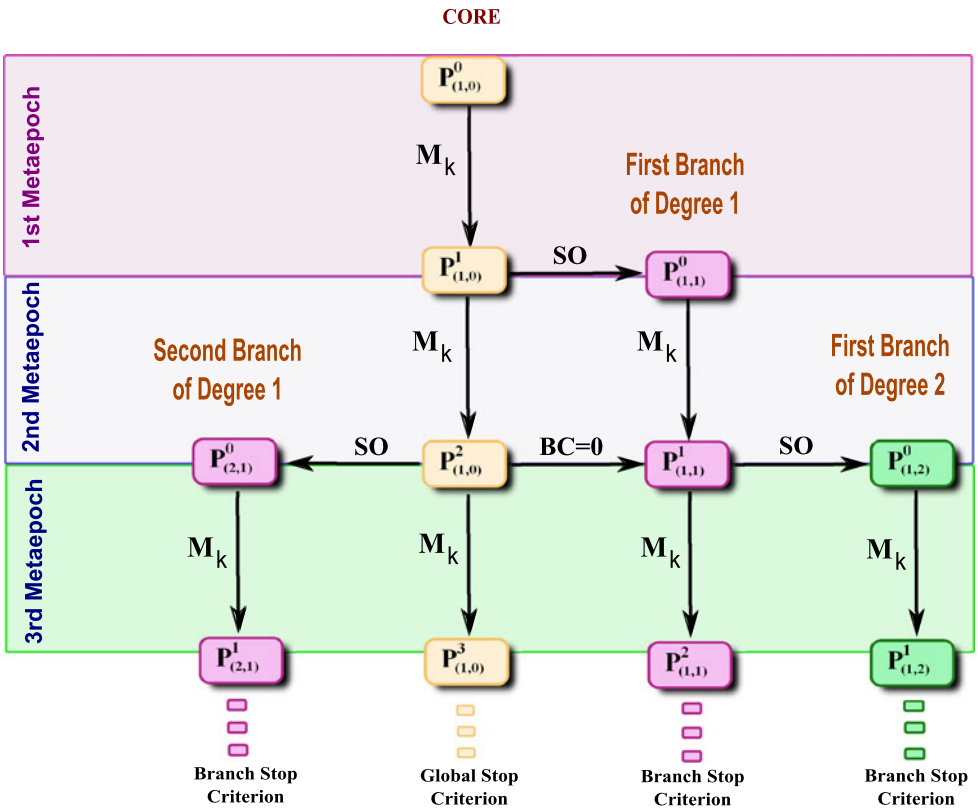


Fig. 4 3 levels of *Green-HGS-Sched* tree structure [27]

determined by the permutations of tasks or machines' labels, but the lengths of these permutation strings may vary as the numbers of tasks and/or machines can change over time. Additional probability distributions should be then specified for an estimation of the system states in considered time intervals.

In this paper we adapted to the green grid scheduling the Hierarchic Genetic Scheduler (*Green-HGS-Sched*) developed in [25] as an effective multi-level hierarchic alternative for the single-population genetic-based schedulers. The main aim of *Green-HGS-Sched* is a comprehensive exploration of the scheduling landscape by the execution of many dependent evolutionary processes. This scheduler can be modeled as a multi-level decision tree. The search process starts by activating a scheduler with the lowest possible accuracy of search, that is interpreted as the "core" in the tree model. This process is responsible for the management of the whole search process, and for the detection of promising partial solutions. More accurate processes are activated in the neighborhoods of those partial solutions for the prevention of the premature convergence of the scheduler and for a possible improvement of the best solutions found in the system. The activation of these processes does not increase significantly the complexity of the hierarchic scheduler because of three main reasons: (i) differently to the hybrid strategies, where the components are usually composed of various meta-heuristics and local search methods, we use the same general framework for the algorithms working at all levels of the tree; (ii) the tree extension is steered by the specialized operations responsible for the deactivation of the ineffective processes and by the effectiveness of the search in the core of the tree; (iii) finally, the synchronization of the search is provided 'horizontally' at each level of the tree, so there is no need to refer to the parental nodes and enables an easy adaptation to the actual system state. For all these reasons *Green-HGS-Sched* significantly differs from the existed hierarchical, hybrid and branching schedulers applied to the various grid scheduling problems and classical job-shop problems (see e.g. [8]).

We present in Fig. 4 an example of 3-level *Green-HGS-Sched* tree structure [27].

Each branch of the tree is created by an active genetic algorithm designed for solving the scheduling problems in CGs. The accuracy of search in *Green-HGS-Sched* branches is defined by the *degree* parameter with lowest value 0 set for the core of the system.

We denote by $P^{e}_{(r,t)}$ a population evolving in the branch of degree t, where:

- $e \in \mathbb{N}$ defines the global metaepoch counter;
- $t \in \{1, \ldots, M\}$, $M \in \mathbb{N}$ and M is the maximal degree of the branches;
- r is the number of branches of the same degree.

The hierarchical structure of the scheduler is updated periodically after the execution of k-generation evolutionary

processes in each active branch. We call such a process a *k-periodic metaepoch* M_k, $(k \in \mathbb{N})$ and define it in the following way:

$$M_k\left(P^e_{(r,t)}\right) = \left(P^{e+l}_{(r,t)}, \widehat{s}\right) \tag{24}$$

where $\widehat{s}$ is the best adapted individual in the metaepoch, $P^e_{(r,t)}$, $(t \in \{1, \ldots, M\}, M \in \mathbb{N})$.

New branches of the higher degree can be created in neighborhoods of the best adapted individuals found in each active branch by using a *Sprouting Operation (SO)* defined as follows:

$$SO\left(P^e_{(r,t)}\right) = \left(P^e_{(r,t)}, P^0_{(r',t+1)}\right) \tag{25}$$

where $P^e_{(r,t)}$ is a parental branch, and $P^0_{(r+1,t+1)}$ denotes an initial population for a new branch of degree $t+1$. Individuals in this population are selected from an S_t-neighborhood $(1 \le S_t \le n)$ of the best adapted individual $\widehat{S}$ in the parental population $P^e_{(r,t)}$. This neighborhood is created by all possible permutations or reassignments of tasks in $(n - S_t)$-length suffix of $\widehat{S}$. The S_t-length prefix of a given schedule S is generated by using the following operator:

$$A_{(S_t)}(S) = \tilde{S}, \qquad |\tilde{S}| = S_t, \quad S_t \le n \tag{26}$$

where $|\tilde{S}|$ denotes the length of the suffix in the permutation sequence which encodes the schedule S. The values of S_t parameters may be different in branches of the different degrees. In this paper we assume that these parameters can be calculated in the following way:

$$S_t = (suf)^t \cdot n \tag{27}$$

where $suf \in [0, 1]$ is a global strategy parameter called a *neighborhood parameter* and t is the branch degree.

The sprouting operation is conditionally activated depending on the outcome of a *Branch Comparison* (BC) binary operator applied for parental and its all directly sprouted branches. It is used for the detection of 'similarity' of the resulting populations in each parental-sprouted pair of branches. Formally the $BC : Q \to \{0, 1\}$ operator is defined by the following formula:

$$BC(X, Y, S_t) = \begin{cases} 1, & \exists x \in X, \exists y \in Y : A_{S_t}(x) = A_{S_t}(y) \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

where $Q = \{(X, Y, S_t)\}$ and X, Y—are the populations in branches of degrees t and $t+1$ respectively. This operator is activated after execution of at least two metaepochs in the core. The outcome of the BC operator is 1 if the parental branch and its "descendant" (sprouted) branch operate in a similar region in the optimization landscape. In such a case another metaepoch is executed in the parental branch without creating any new process. This technique is crucial in an effective management of the algorithm structure by preventing the activation of many similar processes in the same local region, which usually increase significantly the complexity of the whole strategy.

The implementation of the BC operator may be very complex. In our early implementations of *Green-HGS-Sched* we achieved very good results in the minimization of the makespan and other scheduling criteria (see [24]), but the execution time of the scheduler was quite long in more complex grid scenarios. In order to reduce the execution time of the BC procedure, we introduced a hash table with the *task-resource allocation* key denoted by K, that supports the indication of the populations operating in similar regions in the search space. 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 S_t-length suffix in direct representation of the schedule vector (reading the suffix in a circular way). The hash function f_{hash} is defined as follows:

$$f_{\text{hash}}(K) = \begin{cases} 0, & K < K_{\min} \\ \lfloor N \cdot (\frac{K - K_{\min}}{K_{\max} - K_{\min}}) \rfloor, & K_{\min} \le K < K_{\max} \\ N - 1, & K \ge K_{\max} \end{cases} \tag{29}$$

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

In the case of the conditional sprouting of the new branches of the degree $t+1$ from the parental branch of the degree t the keys are calculated for the best individual in the parental branch and individuals in all populations in all active branches of the degree $t+1$. If there is any individual in the higher degree branches, for which the key matches the key of the best adapted individual in the parental branch, then the value of BC is 1 and no branch of the degree $t+1$ is sprouted.

In the case of the comparison of the branches of the same degree t, all branches, in which there exists the individuals with the identical keys have to be reduced and a single joint branch is created (the value of BC is 1). The individuals in this branch are selected from the "youngest" (in the sense of the population evolution) populations in all reduced branches.

It has been shown in [47] and [25] that hash technique can reduce significantly (50–70 %) the execution time of the genetic algorithms, where indication of the similarity of solutions is necessary.

7.1 Genetic Engine in *Green-HGS-Sched*

The main genetic engine in *Green-HGS-Sched* branches is defined in Alg. 1. It is based on the framework of the classical genetic algorithms used in the combinatorial optimization [36].

We apply the *direct representation* of the schedules in the base populations P^t and P^{t+1}, and *permutation representation* in P^t_c and P^t_m populations to implement the crossover

Algorithm 1 A template of the genetic engine for six genetic-based grid schedulers

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

and mutation operators. An initial population is generated randomly. Based on our previous results of implementation of genetic-based meta-heuristics to green scheduling in grids we use the following configuration of genetic operations in the main loop of the Algorithm 1: (i) *Linear Ranking* as selection scheme, (ii) *Cycle Crossover (CX)* operator and (iii) *Move* mutation method [36].

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

We consider two alternate replacement mechanism for the generation of the base population for a new GA loop, namely *Elitist Generational* and *Struggle* strategies. In *Elitist Generational* method the "Elite" of the best solutions contains just 2 individuals. The main drawback of such methods is that they may lead to premature convergence on some solution and impacts on the stagnation of the population. A *Struggle* mechanism can be an effective tool for avoiding too fast scheduler's convergence to the local optima. In this method, new generation of individuals is created by replacing a part of the population by the most similar individuals—if this replacement minimizes the fitness value. The definition of the struggle 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* [32] for measuring the distances between schedules according to the following formula:

$$sim_e(S_1; S_2) = \sqrt{\sum_{j=1}^{n} \frac{(S_1[j] - S_2[j])^2}{\sigma_P^2}} \tag{30}$$

where σ_P is the standard deviation of the $S_1[j]$ over the population P.

The possible high computational cost of the struggle strategy may be reduced by implementing a *hash technique*, as it was proposed in the previous section for BC operator. Using the struggle replacement mechanism in genetic grid schedulers allow us a fine tuning of the scheduler to "converge" to a good solution depending on available time (for instance, scheduler's time activation interval) [47].

8 Empirical evaluation of the hierarchical genetic schedulers

In this section we present the results of empirical evaluation of two implementation of *Green-HGS-Sched*, namely *HGS-Elit* and *HGS-St* with Elitist Generational and Struggle replacement mechanisms (respectively) in the branches. We compare the efficiency of hierarchical schedulers with the results achieved by single-population GAs and Island Models with the same configuration of genetic operators and parameters. The experiments were conducted in two "energetic" scenarios, namely *Max-Min Mode* and *Modular Power Supply Mode* defined in Sect. 6.2. For simulating various grid size scenarios in static and dynamic modes we used the *Energy-aware Hyper-G* grid simulator introduced in [26]. The main idea of the simulator is presented in Sect. 8.1. The empirical results are analyzed in Sect. 8.3.

8.1 Energy-aware HyperSim-G grid simulator

Energy-aware HyperSim-G simulator is an extension of the HyperSim-G software [48] dedicated s for modeling the realistic CGs systems in various energetic scenarios. In energy-aware scheduling the instance contains the following input data:

- workload vector of tasks;
- computing capacities of machines;
- prior loads of machines;
- 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.);
- DSVF levels matrix for machine categories; and
- the ETC matrix.

The input data is needed for generation of the scheduling event, which is passed on to the selected scheduler in order to compute the optimal schedule(-s). Finally, the scheduler sends the optimal schedule(-s) back to the simulator, which allocate the resources and simulate the computation process.

The instances produced by the simulator for our experiments are divided into static and dynamic grid scheduling benchmarks. In the static case, the number of tasks and

Table 2 Values of key parameters of the grid simulator in static and dynamic cases

	Small	Medium	Large	Very large
Static case				
Nb. of hosts	32	64	128	256
Resource cap. (in MHz CPU)	$N(5000, 875)$			
Total nb. of tasks	512	1024	2048	4096
Workload of tasks	$N(250000000, 43750000)$			
Dynamic case				
Init. hosts	32	64	128	256
Max. hosts	37	70	135	264
Min. hosts	27	58	121	248
Resource cap. (in MHz CPU)	$N(5000, 875)$			
Add host	$N(625000, 93750)$	$N(562500, 84375)$	$N(500000, 75000)$	$N(437500, 65625)$
Delete host	$N(625000, 93750)$			
Total tasks	512	1024	2048	4096
Init. tasks	384	768	1536	3072
Workload	$N(250000000, 43750000)$			
Interarrival	$E(7812, 5)$	$E(3906, 25)$	$E(1953, 125)$	$E(976, 5625)$

the number of machines remain constant during the simulation, while in the dynamic case, both parameters may vary over time. In both static and dynamic cases four Grid size scenarios are considered: (a) small (32 hosts/512 tasks), (b) medium (64 hosts/1024 tasks), (c) large (128 hosts/2048 tasks), and (d) very large (256 hosts/4096 tasks).

The simulator is highly parameterized in order to reflect the realistic grid scenarios. The main parameters are defined as follows:

- *Number of hosts*: Number of resources in grid;
- *MIPS*: A probability distribution specified for modeling the computing capacity of resources;
- *Total tasks*: Number of tasks in a given batch;
- *Workload*: A probability distribution used for modeling the workload of tasks;
- *Host selection*: Selection policy of resources, the parameter (*all* means that all resources of the system are selected for scheduling purposes);
- *Task selection*: Selection policy of tasks, the parameter (*all* means that all tasks in the system must be scheduled);
- *Number of runs*: Number of simulations done with the same parameters, reported results are then averaged over this number.

In Table 2 we present the key input parameters of the simulator in the static and dynamic cases. We use the notation $N(a, b)$ and $E(c, d)$ for Gaussian and exponential probability distributions. We have used in the experiments the similar settings for our simulator as in our previous work [26, 29]. These parameters were tuned for illustrating the typical grid size scenarios for conventional grid scheduling in [48].

In the dynamic case we have to specify the minimal and maximal values for numbers of tasks and machines in the system. The resources can be dropped or added to grid with the frequencies defined by the Gaussian distributions (*add host* and *delete host* parameters). New tasks may arrive in the system with frequency parameter denoted by *interarrival*, until a *total tasks* value is reached. An *Activation* parameter establishes the activation policy according to an exponential distribution. The already scheduled tasks that have not been executed yet will be rescheduled if *reschedule* is true.

We consider 16 DVFS levels for three "energetic" resource classes: Class I, Class II and Class III presented in Table 1.

8.2 Scheduling meta-heuristics and performance measures

We consider in experiments six genetic-based meta-heuristics defined three genetic schedulers defined in Table 3.

Table 3 Six GA-based grid schedulers evaluated in the experimental analysis

Scheduler	Type of algorithm	Replacement method
GA-Elit	Single-population GA	Elitist Generational
GA-St	Single-population GA	Struggle
IGA-Elit	Island GA	Elitist Generational
IGA-St	Island GA	Struggle
HGS-Elit	*Green-HGS-Sched*	Elitist Generational
HGS-St	*Green-HGS-Sched*	Struggle

The aforementioned methodologies differ in the implementation of the replacement mechanism in the main genetic framework. We used *Elitist Generational* replacement in *xxx-Elit* algorithms and *Struggle* procedure in *xxx-St* algorithms. Both single-population GAs, namely *GA-Elit* and *GA-St*, are implemented as the main genetic mechanism in *IGA-Elit*, *HGS-Elit*, *IGA-St* and *HGS-St* respectively.

Island Genetic Algorithm (IGA) [46] is a well-known parallel GA technique. An initial population (possibly big) 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 separate algorithm for each deme). After fixed number of iterations (we denote it by it_d) the migration procedure is activated, which enables the partial exchange (usually according to the standard ring topology) of the individuals among islands. The relative amount of the migrating individuals denoted by mig, is the algorithm global parameter called a *migration rate*. It is calculated in the following way:

$$mig = \frac{m_{\text{deme}}}{\text{deme}} \times 100\ \%, \tag{31}$$

where *deme* is the size of the sub-population in IGA and m_{deme} is the number of migrating individuals in each deme.

In Tables 4, 5 and 6 we present the configuration of the key parameters for both implementations of single-population GA, IGA and *Green-HGS-Sched* meta-heuristics respectively. The size of initial and intermediate populations in IGA depends on the implementation of the genetic engine in islands and are the same as for single-population *GA-Elit* and *GA-St* algorithms. Similarly as in the case of the parametrization of the grid simulator, we based on the configuration of the conventional implementation of *HGS-Sched* scheduler and single-population genetic grid schedulers presented in [26, 29] and [47], where the detailed tuning process has been provided.

Table 4 GA setting for static and dynamic benchmarks

Parameter	GA-Elit	GA-St
Evolution steps	$5 * m$	$20 * m$
Pop. size (*pop_size*)	$\lceil(\log_2(m))^2 - \log_2(m)\rceil$	$4 * (\log_2(m) - 1)$
Intermediate pop.	$pop\_size - 2$	$(pop\_size)/3$
Cross probab.	1.0	1.0
Mutation probab.	0.2	
max_time_to_spend	30 secs (*static*)/45 s (*dynamic*)	

The relative performance of all six schedulers is measured through the two following metrics:

- minimal makespan defined as follows:

$$makespan = \min\{Makespan_I, Makespan_{II}\} \tag{32}$$

- a relative energy consumption improvement rate expressed as follows:

$$\text{Im}(E) = \frac{E_I - E_{\text{batch}}}{E_{\text{batch}}} \times 100\ \%, \tag{33}$$

where E_{batch} and E_I are defined in Eqs. (10) and (20) respectively.

8.3 Results

Each experiment was repeated 30 times under the same configuration of operators and parameters. In Figs. 5 and 6 we present the box-plots of the makespan values for six considered schedulers (confidence level—95 %). The makespan is measured and expressed in arbitrary time units, the same as defined in ETC matrix model (see Sect. 4).

Both implementations of *Green-HGS-Sched* achieved the best results in all instances but Large grid in static case and Small and Large instances in the dynamic case, in which they lose to IGA model. The results of a simple comparison of the impact of the replacement method on the algorithms' performance provided for all pairs of the *xxx-Elit* and *xxx-St* schedulers show that Struggle replacement is much more effective that Elitist Generational method in the case of single-population GA and IGA schedulers. It confirms the results

Table 5 *Green-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 s (*static*)/70 s (*dynamic*)

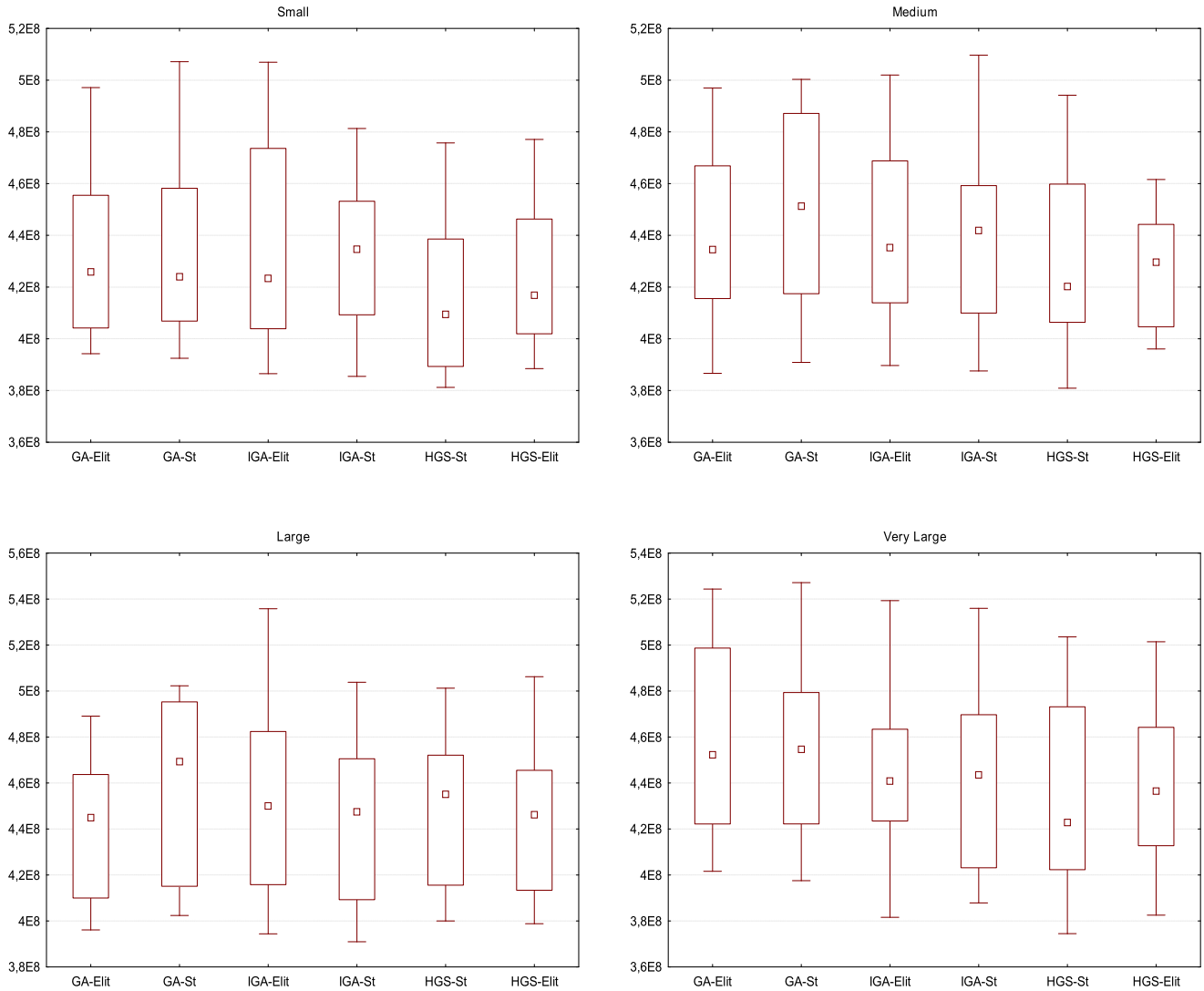


Fig. 5 The box-plot of the results for makespan in static case

Table 6 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 s (*static*)/70 s (*dynamic*)

of our preliminary study on the effectiveness of single-population genetic schedulers in CGs presented in [26]. For *Green-HGS-Sched* the situation is completely different. In most of the cases the effectiveness of both hierarchical implementations are at the comparative levels, with a little advantage of elite technique in the dynamic case. It means that in *Green-HGS-Sched* the most important is the fast exploration by the core of the system of probably wider regions in the search space than in the case of GA and IGA implementations. The core can activate the more accurate processes in the neighborhoods of the partial solutions which are undetected by the other schedulers, which makes the *Green-HGS-Sched* very effective in the exploration of new regions in the optimization domain and in escaping the basins of attraction of the local solutions. The complexity of the hierarchic system is in fact not a drawback of the scheduler, cause the constraints of the execution time for HGS and IGA are exactly the same. The ranges in the achieved makespan values for all considered meta-heuristics are not greater than 30–45 % of the mean makespan values, which means that the stability of all schedulers in all cases are acceptable. The distributions of the makespan results are asymmetric: the skewness in the static case is positive, for GA and IGA

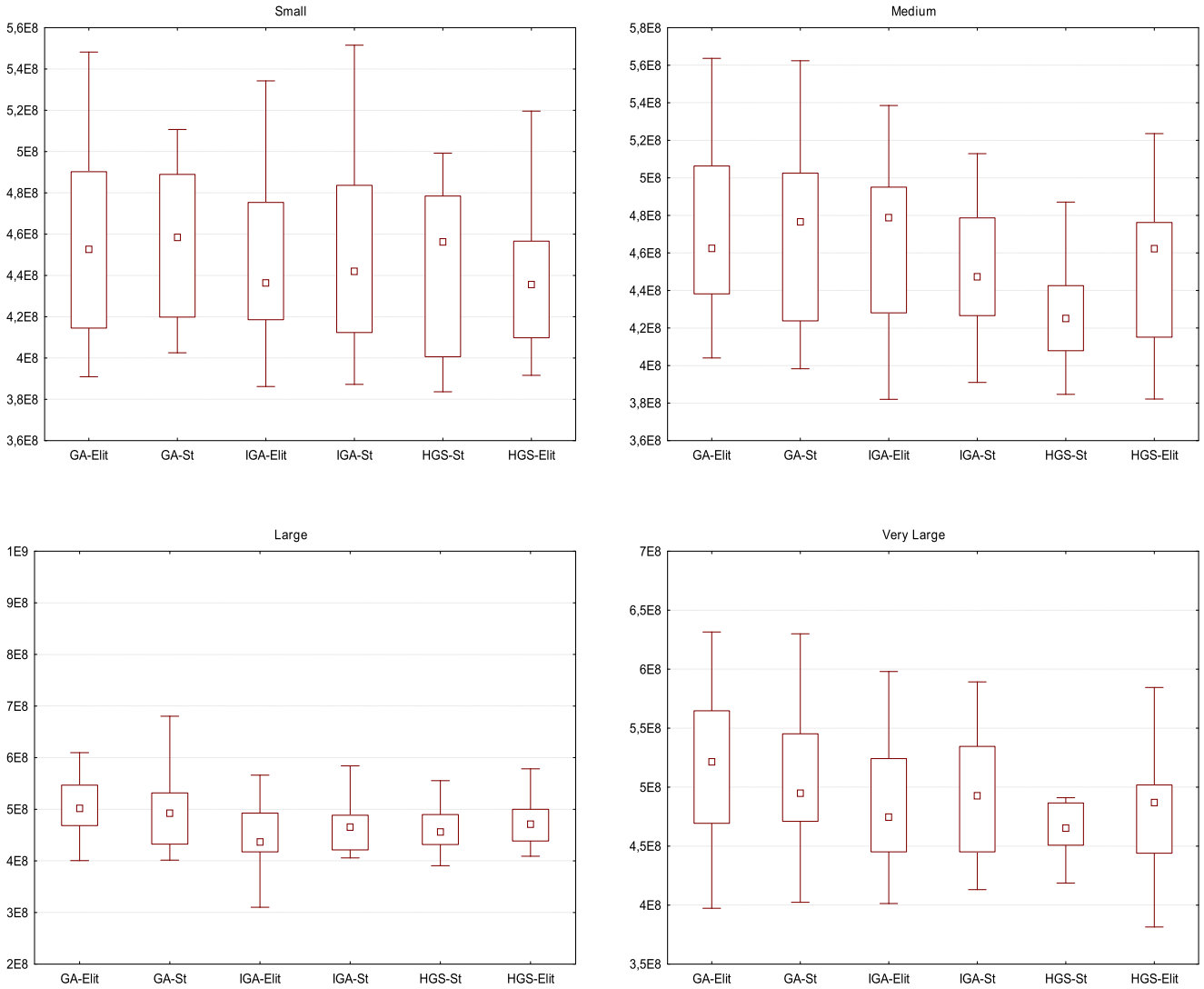


Fig. 6 The box-plot of the results for makespan in dynamic case

and negative for *Green-HGS-Sched* in most of the static instances, however it is negative in the dynamic grids for almost all schedulers. It means that the reduction of the average makespan in this case is much harder than in static case (the mean values are closer the third quantile, than the first one), which confirms the complexity of the problem in the realistic dynamic grid scenarios.

The box-plots for the energy saving rates $\mathrm{Im}(E)$ are presented in Figs. 7 and 8.

The results of the energy optimization differs significantly compare with the makespan results. In this case each of *IGA-Elit* and *GA-Elit* algorithms outperforms the rest schedulers in four instances. *Green-HGS-Sched* is not as good in energy optimization as in makespan minimizing. It means that it works quite good in Min-Max scenario, so no additional DSV modules are necessary here. The range of the average saving rate values is 10 %–35 % for most of the schedulers, which is rather high. Finally, it can be observed that the skewness of the distribution of the results is positive or neutral for the worst "energy optimizers" and negative for the best ones.

9 Conclusions

We addressed in this paper the problem of optimizing the energy utilized in CGs in independent batch scheduling. Our energy management model is based on *Dynamic Voltage Scaling (DVFS)* technique adapted to the dynamic grid environment. We formalized the grid scheduling problem as a bi-objective optimization task with makespan and average energy consumption as the main objectives.

For solving the addressed grid scheduling problem, we developed two implementations of an energy-efficient Hierarchical Grid Scheduler *Green-HGS-Sched* and provided

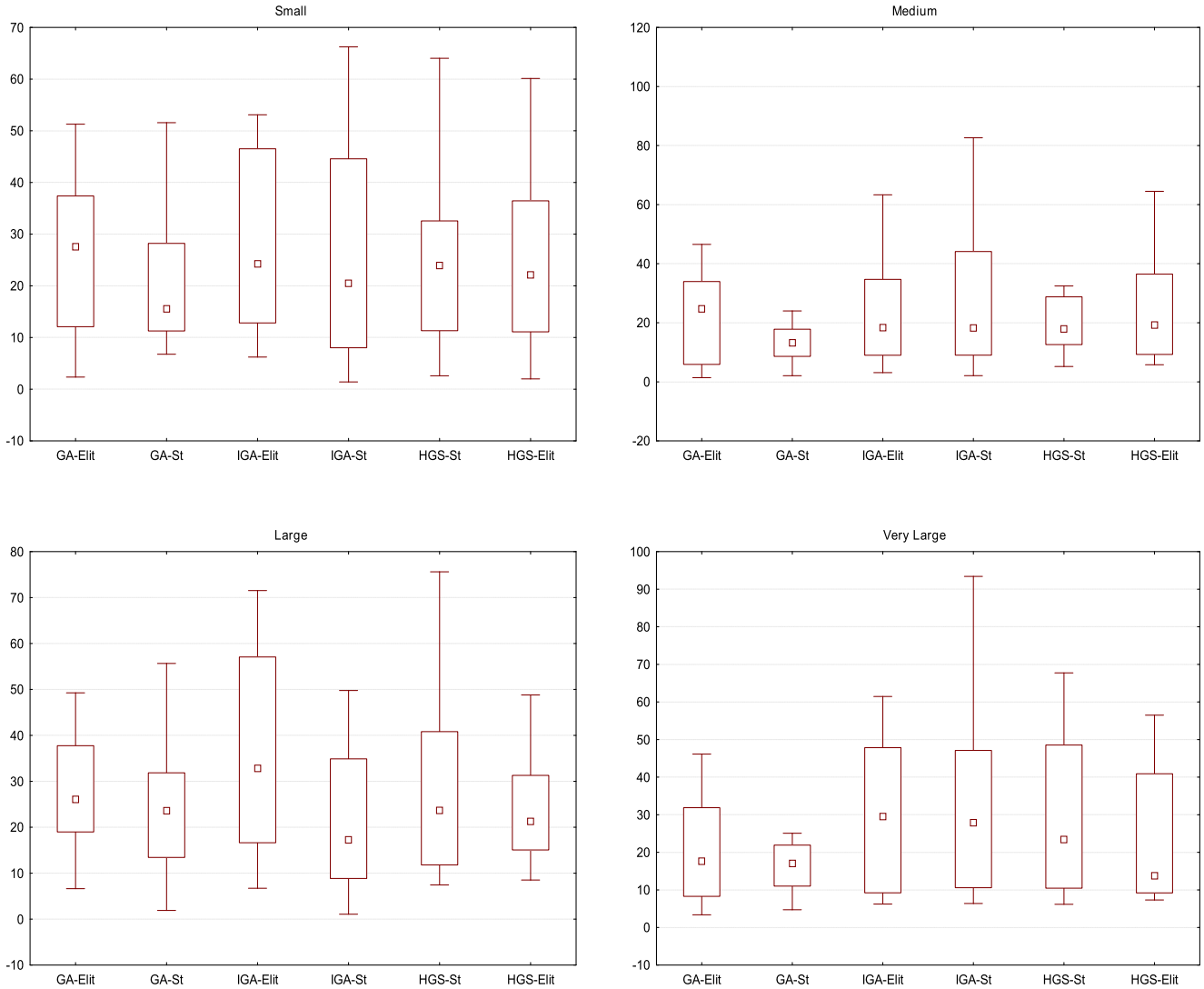


Fig. 7 The box-plot of the results for relative energy saving rate in static case (in %)

its experimental evaluation in two 'energetic' scheduling modes in static and dynamic grid scenarios under the makespan and relative energy consumption improvement rate. Their effectiveness were compared with the results achieved by four single-population Genetic Algorithm (GA) and Island GA schedulers. To provide the experiments, we integrated all energy-aware schedulers within a grid simulator. The simulation results confirmed the effectiveness of the proposed schedulers in the reduction of the energy consumed by the whole system and in dynamic load balancing of the resources in grid clusters, which is sufficient to maintain the desired quality level(-s).

Our model is general in its implementation and can be easily adapted to a particular scenario and realistic grid infrastructure, such as the large-scale banking system or highly distributed data system. First, we do not consider any special architectures for grid resources, which means that this characteristics can be specify separately and integrated with the system simulator. The term "task" can be also used for monolithic applications, metatasks or parallel applications represented by *Directed Acyclic Graphs*. The schedulers are integrated with the main grid simulator as separate modules, and therefore they can be easily modified, extended and hybridized with the other algorithms. Finally, we simulate the dynamics of the realistic grid system, in which the availability of the resources and the number of tasks may vary over the time.

Acknowledgements Dr. Kołodziej's and Dr. Byrski's research presented here was partially supported by "Biologically inspired mechanisms in planning and management of dynamic environments" grant of Polish National Science Center, No. N N516 500039.

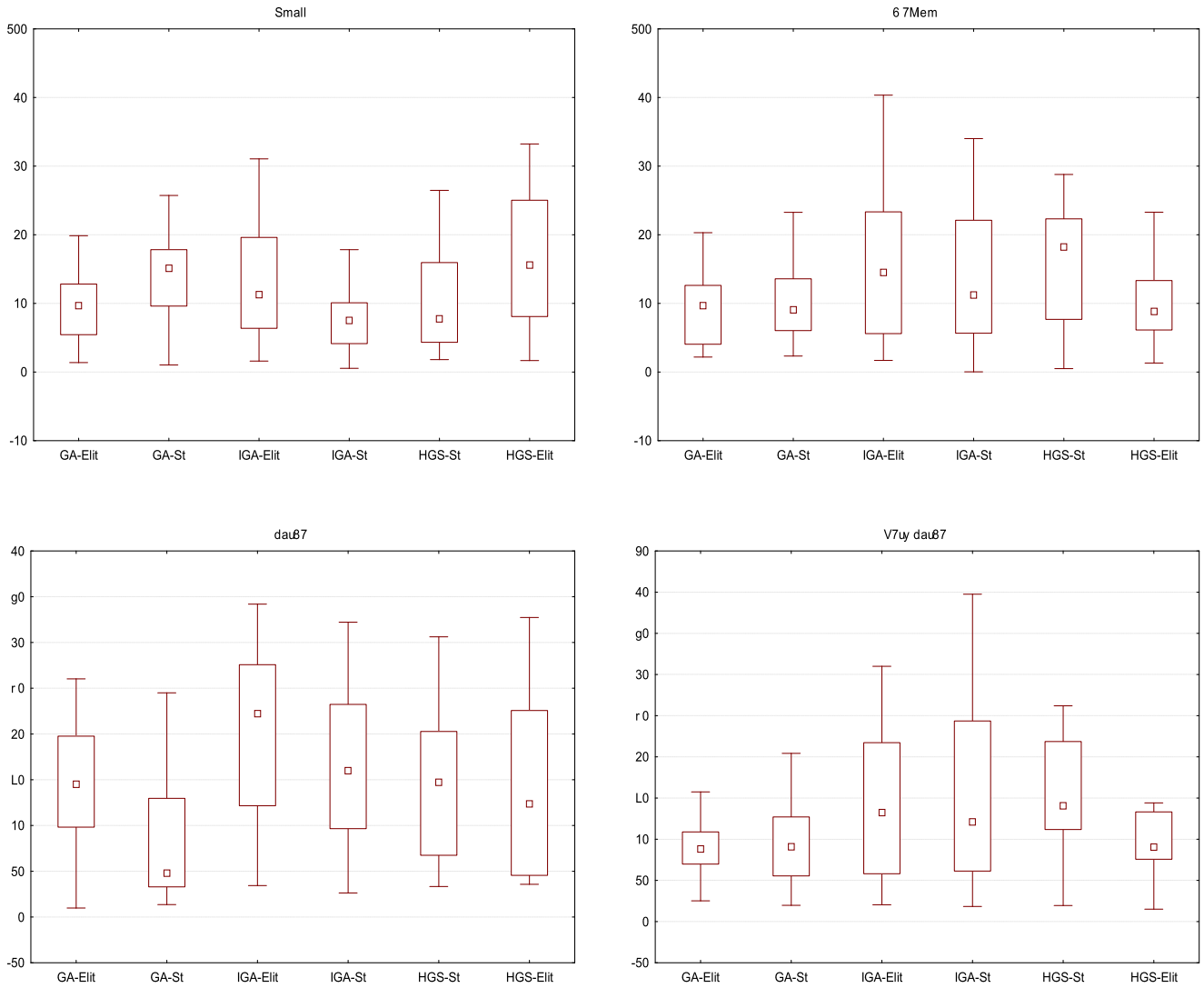


Fig. 8 The box-plot of the results for relative energy saving rate in dynamic case (in %)

References

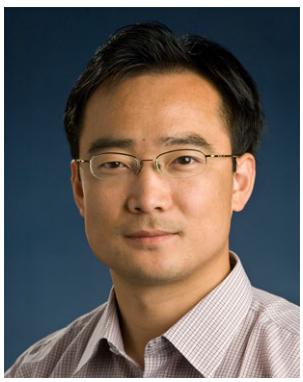
1. Abraham, A., Buyya, R., Nath, B.: Nature's heuristics for scheduling jobs on computational grids. In: Proc. of the 8th IEEE ACC, India (2000)
2. Ali, S., Siegel, H.J., Maheswaran, M., Hensgen, D.: Task execution time modeling for heterogeneous computing systems. In: Proceedings of Heterogeneous Computing Workshop, pp. 185–199 (2000)
3. Andersen, D.G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., Vasudevan, V.: FAWN: a fast array of wimpy nodes. In: 22nd ACM Symposium on Operating Systems Principles (SOSP). Big Sky, MT (2009)
4. Baker, R.J.: CMOS: Circuit Design, Layout, and Simulation, 2nd edn. Wiley, New York (2008)
5. Beloglazov, A., Buyya, R., Lee, Y.C., Zomaya, A.Y.: A taxonomy and survey of energy-efficient data centers and cloud computing systems. Adv. Comput. **82**, 47–111 (2011)
6. Błażewicz, J., Brauner, N., Finke, G.: Scheduling with discrete resource constraints. In: Leung, J.Y.-T. (ed.) Handbook of Scheduling, Chap. 23, pp. 23.1–23.18. Chapman & Hall/CRC, London (2004)
7. Błażewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G.: Scheduling subject to resource constraints: classification and complexity. Discrete Appl. Math. **5**(1), 11–24 (1983)
8. Brucker, P.: Scheduling Algorithms. Springer, Berlin (2007)
9. Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: notation, classification, models, and methods. Eur. J. Oper. Res. **112**(1), 3–41 (1999)
10. Caulfield, A.M., Grupp, L.M., Swanson, S.: Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In: 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), p. 09 (2009)

11. Cohen, J., Cordeiro, D., Trystram, D., Wagner, F.: Multi-organization scheduling approximation algorithms. Concurr. Comput. **23**(17), 2220–2234 (2011)
12. Enokido, T., Aikebaier, A., Takizawa, M.: Process allocation algorithms for saving power consumption in peer-to-peer systems. IEEE Trans. Ind. Electron. **58**(6), 2097–2105 (2011)
13. Fernandez-Baca, D.: Allocating modules to processors in a distributed system. IEEE Trans. Softw. Eng. **15**(11), 1427–1436 (1989)
14. Fibich, P., Matyska, L., Rudová, H.: Model of grid scheduling problem. In: Proc. of the AAAI-05 Workshop on Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing (2005)
15. Garg, S.K., Yeo, C.S., Anandasivam, A., Buyya, R.: Energy-efficient scheduling of HPC applications in cloud computing environments. CoRR abs/0909.1146 (2009)
16. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math. **5**, 287–326 (1979)
17. Guzek, K., Pecero, J.E., Dorrosoro, B., Bouvry, P., Khan, S.U.: A cellular genetic algorithm for scheduling applications and energy-aware communication optimization. In: ACM/IEEE/IFIP International Conference on High Performance Computing and Simulation (HPCS). Cane, France, June 2010, pp. 241–248 (2010)
18. Hotovy, S.: Workload evolution on the Cornell theory center IBM SP2. In: Job Scheduling Strategies for Parallel Proc. Workshop (IPPS'96), pp. 27–40 (1996)
19. Kessaci, Y., Mezmaz, M., Melab, N., Talbi, E.-G., Tuyttens, D.: Parallel evolutionary algorithms for energy aware scheduling. In: Bouvry, P., Gonzalez-Velez, H., Kołodziej, J. (eds.) Intelligent Decisions Systems in Large-Scale Distributed Environments, Studies in Computational Intelligence Series, vol. 362, Chap. 4, pp. 75–100. Springer, Berlin (2011)
20. Khan, S.U.: A self-adaptive weighted sum technique for the joint optimization of performance and power consumption in data centers. In: 22nd International Conference on Parallel and Distributed Computing and Communication Systems (PDCCS), USA, pp. 13–18 (2009)
21. Khan, S.U., Ahmad, I.: A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids. IEEE Trans. Parallel Distrib. Syst. **20**(3), 346–360 (2009)
22. Kliazovich, D., Bouvry, P., Khan, S.U.: DENS: data center energy-efficient network-aware scheduling. In: ACM/IEEE International Conference on Green Computing and Communications (GreenCom), Hangzhou, China, pp. 69–75 (2010)
23. Klusaček, D., Rudová, H.: Efficient grid scheduling through the incremental schedule-based approach. Comput. Intell. **27**(1), 4–22 (2011)
24. Kołodziej, J., Xhafa, F., Kolanko, Ł.: Hierarchic genetic scheduler of independent jobs in computational grid environment. In: Otamendi, J., Bargieła, A., Montes, J.L., Doncel Pedrera, L.M. (eds.) Proc. of 23rd ECMS, Madrid, 9–12.06.2009, Dudweiler, Germany, pp. 108–115. IEEE Press, New York (2009)
25. Kołodziej, J., Xhafa, F.: Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population. Future Gener. Comput. Syst. **27**, 1035–1046 (2011), doi:10.1016/j.future.2011.04.011
26. Kołodziej, J., Khan, S.U., Xhafa, F.: Genetic algorithms for energy-aware scheduling in computational grids. In: Proc. of the 6th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC2011) (2011)
27. Kołodziej, J.: Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems, Studies in Computational Intelligence, vol. 419, Springer, Berlin (2012)
28. Kołodziej, J., Khan, S.U., Zomaya, A.Y.: A taxonomy of evolutionary-inspired solutions for energy optimization: problems and intelligent resolution techniques. In: Kołodziej, J., Khan, S.U., Burczyński, T. (eds.) Advances in Intelligent Modelling and Simulation: Artificial Intelligence-based Models and Techniques in Scalable Computing, Chap. 10, Studies in Computational Intelligence, vol. 422. Springer, Berlin (2012)
29. Kołodziej, J., Khan, S.U., Wang, L., Zomaya, A.Y.: Energy efficient genetic-based schedulers in computational grids. Concur. Comput. doi:10.1002/cpe.2839, 2012
30. Lee, Y.C., Zomaya, A.Y.: Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), China, Shanghai, pp. 92–99 (2009)
31. Lorch, J.R., Smith, A.J.: Improving dynamic voltage scaling algorithms with pace. In: 2001 ACM SIGMETRICS international conference on measurement and modeling of computer systems, pp. 50–61 (2001)
32. Mann, P.S.: Introductory Statistics, 7th edn. Wiley, New York (2010)
33. Mejia-Alvarez, P., Levner, E., Mossé, D.: Adaptive scheduling server for power-aware real-time tasks. ACM Trans. Embed. Comput. Syst. **3**(2), 284–306 (2004)
34. Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y.C., Talbi, E.-G., Zomaya, A.Y., Tuyttens, D.: 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 (2011)
35. Miao, L., Qi, Y., Hou, D., Dai, Y.H., Shi, Y.: A multi-objective hybrid genetic algorithm for energy saving task scheduling in CMP system. In: Proc. of IEEE Intl. Conf. on Systems, Man and Cybernetics (ICSMC2008), pp. 197–201. doi:10.1109/ICSMC.2008.4811274 (2008)
36. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin (1992)
37. Milaneś-Montero, M.I., Romero-Cadaval, E., Barrero-González, F.: Hybrid multiconverter conditioner topology for high-power applications. IEEE Trans. Ind. Electron. **58**(6), 2283–2292 (2011)
38. Min, R., Furrer, T., Chandrakasan, A.: Dynamic voltage scaling techniques for distributed microsensor networks. In: Proc. IEEE Workshop on VLSI, pp. 43–46 (2000)
39. Pinel, F., Pecero, J., Bouvry, P., Khan, S.U.: Memory-aware green scheduling on multi-core processors. In: 39th IEEE International Conference on Parallel Processing (ICPP). pp. 485–488, USA (2010)
40. Rahman, M., Ranjan, R., Buyya, R., Benatallah, B.: A taxonomy and survey on autonomic management of applications in grid computing environments. Concurr. Comput. **23**(16), 1990–2019 (2011)
41. Shen, G., Zhang, Y.Q.: A new evolutionary algorithm using shadow price guided operators. Appl. Soft Comput. **11**(2), 1983–1992 (2011)
42. Shen, G., Zhang, Y.Q.: 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 (2011)
43. Valentini, G.L., Lassonde, W., Khan, S.U., Min-Allah, N., Madani, S.A., Li, J., Zhang, L., Wang, L., Ghani, N., Kołodziej, J., Li, H., Zomaya, A.Y., Xu, C.-Z., Balaji, P., Vishnu, A., Pinel, F., Pecero, J.E., Kliazovich, D., Bouvry, P.: An overview of energy efficiency techniques in cluster computing systems. Clust. Comput. doi:10.1007/s10586-011-0171-x (2011)
44. Wang, L., Khan, S.U.: Review of performance metrics for green data centers: a taxonomy study. J. Supercomput., 1–1. doi:10.1007/s11227-011-0704-3 (2011)
45. Warren, M.S., Weigle, E.H., Feng, W.-C.: High-density computing: a 240-processor Beowulf in one cubic meter. In: Proc. of the IEEE/ACM SC2002 Conference, Baltimore, Maryland (2002)

46. Whitley, D., Rana, S., Heckendorn, R.B.: The island model genetic algorithm: on separability, population size and convergence. CIT, J. Comput. Inf. Technol. **7**, 33–47 (1998)
47. Xhafa, F., Carretero, J., Abraham, A.: Genetic algorithm based schedulers for grid computing systems. Int. J. Innov. Comput., Inf. Control **3**(5), 1053–1071 (2007)
48. Xhafa, F., Carretero, J., Barolli, L., Durresi, A.: Requirements for an event-based simulation package for grid systems. J. Intercon. Netw. **8**(2), 163–178 (2007)
49. Zomaya, A.Y.: 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 (2009)

Joanna Kołodziej graduated in Mathematics from the Jagiellonian University in Cracow in 1992, where she also obtained the Ph.D. in Computer Science in 2004. She is employed at Cracow University of Technology as an Assistant Professor. She has served and is currently serving as PC Co-Chair, General Co-Chair and IPC member of several international conferences and workshops including PPSN 2010, ECMS 2011, CISIS 2011, 3PGCIC 2011, CISSE 2006, CEC 2008, IACS 2008–2009, ICAART 2009–2010. Dr. Kołodziej is Managing Editor of IJSSC Journal and serves as a EB member and guest editor of several peer-reviewed international journals. For more information, please visit: http://www.joannakolodziej.org/.

Samee Ullah Khan is Assistant Professor of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA. Prof. Khan has extensively worked on the general topic of resource allocation in autonomous heterogeneous distributed computing systems. As of recent, he has been actively conducting cuttingedge research on energy-efficient computations and communications. A total of 111 (journal: 40, conference: 51, book chapter: 12, editorial: 5, technical report: 3) publications are attributed to his name. For more information, please visit: http://sameekhan.org/.



Lizhe Wang is a full professor at Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, China. He also holds a "ChuTian Scholar" Chair Professor position at School of Computer, China University of Geosciences. Dr. Wang received his Bachelor and Master of Engineering degrees from Tsinghua University, China and Doctoral of Engineering from University Karlsruhe, Germany. His research interests include high performance computing, data-intensive computing, Grid/Cloud computing.

Aleksander Byrski obtained his Ph.D. in 2007 at AGH University of Science and Technology in Cracow. He works as an assistant professor at the Department of Computer Science of AGH University of Science and Technology. His research focuses on multi-agent systems, biologically-inspired computing and other soft computing methods.

Nasro Min-Allah received his Undergraduate and Master degrees in Electronics and Information Technology in 1998 and 2001 respectively. He obtained a Ph.D. in real-time systems from the graduate university of the Chinese Academy of Sciences, P.R. China in 2008. Currently, he is working as Associate Professor at the Department of Computer Science, COMSATS Institute of Information Technology (CIIT), Islamabad, Pakistan. He is heading the Department of Computer Science, CIIT – Islamabad since February 2010. His main research is focused on scheduling theory, green computing, cloud computing and fault tolerant real-time systems.

Sajjad Ahmad Madani works at COMSATS institute of information technology (CIIT) Abbottabad Campus as associate professor. He joined CIIT in August 2008 as Assistant Professor. Previous to that, he was with the institute of computer technology from 2005 to 2008 as guest researcher where he did his Ph.D. research. He has already done B.Sc. Civil Engineering from UET Peshawar and was awarded a gold medal for his outstanding performance in academics. His areas of interest include low power wireless sensor network and application of industrial informatics to electrical energy networks. He has published more than 35 papers in international conferences and journals.