Cluster Comput
DOI 10.1007/s10586-015-0437-9

Power-aware resource allocation in computer clusters using dynamic threshold voltage scaling and dynamic voltage scaling: comparison and analysis

Kashif Bilal · Ahmad Fayyaz · Samee U. Khan · Saeeda Usman

Received: 5 June 2014 / Revised: 10 November 2014 / Accepted: 22 January 2015

Abstract One of the major challenges in the high performance computing (HPC) clusters is intelligent power management to improve energy efficiency. The key contribution of the presented work is the modeling of a Power Aware Job Scheduler (PAJS) for HPC clusters, such that the: (a) threshold voltage is adjusted judiciously to achieve energy efficiency and (b) response time is minimized by scaling the supply voltage. The PAJS considers the symbiotic relationship between power and performance and caters the optimization of the both, simultaneously. The key novelty in our work is utilization of the dynamic threshold-voltage scaling (DTVS) for the reduction of cumulative power utilized by each node in the cluster. Moreover, to enhance the performance of the resource scheduling strategies in this work, independent tasks within a job are scheduled to most suitable computing nodes (CNs). This paper analyzes and compares eight scheduling techniques in terms of energy consumption and makespan. Primarily, the most suitable dynamic voltage scaling (DVS) level adhering to the deadline is identified for each of the CNs by the scheduling heuristics. Afterwards, the DTVS is employed to scale down the static, as well as dynamic power by regulating the supply and bias voltages. Finally, the per node threshold scaling is used attain power saving. Our simulation results affirm that the proposed methodology significantly reduces the energy consumption using the DTVS.

Keywords Resource allocation · Energy efficiency · Distributed systems · Cloud computing

K. Bilal (✉)
Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan
e-mail: Kashifbilal@ciit.net.pk

A. Fayyaz · S. U. Khan · S. Usman
Electrical and Computer Engineering Department, North Dakota State University, Fargo, ND, USA
e-mail: ahmad.fayyaz@ndsu.edu

S. U. Khan
e-mail: samee.khan@ndsu.edu

S. Usman
e-mail: saeeda.usman@ndsu.edu

1 Introduction

With the enormous progression in computer technology, the need for power awareness has rapidly increased. Whether it's supercomputing center, cluster computing, or a large-scale data center, the minimization of energy consumption is a serious concern [31]. Implementation of energy efficient workstations is an indispensable need of the day due to the rising energy cost and environmental impacts. The demand for the reduction in energy consumption is even higher in large clusters, because the annual power budget of the cluster is approximately equal to the cost of a new server [11].

Cluster computing can be defined as a single system image of multiple computing resources combined together through networks, software, and hardware to handle complex computations [31]. The high performance computing (HPC) clusters are widely used to render remarkable computing capabilities for scientific, as well as commercial applications [30]. Rigorous and complex research problems, such as complex image generation, molecular level design, and weather modeling can be solved using clusters, super computers, distributed computers, cloud, and grids [32]. The need for efficient processing of the aforementioned tasks has escalated the demand of cluster deployment to a significant level. However, despite the benefits cluster computing offers, a key challenge is the reduction in energy consumption. Energy consumption of data centers, grids, and computer clusters is getting doubled almost every five years (since last 15 years). It is esti-

Published online: 08 February 2015

mated that almost 50 % of the operational expenses within a data center accounts for the energy cost [1,31].

This paper presents an empirical approach to reduce response time and energy consumption while maintaining high performance using resource scheduling algorithms. The dynamic threshold-voltage scaling (DTVS) and dynamic voltage scaling (DVS) are the two major techniques employed to minimize the power budget of a large scale cluster [25]. In the DVS, the supply voltage, V_{DD}, is scaled down to a discrete number of voltage levels without violating the task's deadline for energy saving. Alternatively, the DTVS manages both the leakage and dynamic power by adjusting the V_{DD} and bias voltage (V_{BS}) simultaneously, to improve power saving at increased activity levels. The operational voltage of the computing nodes (CNs) can be decreased by employing DTVS.

Previous works [15,26] focuses solely on exploiting DVS to achieve energy savings. However, the work presented here hinges on introducing DTVS along with DVS to further minimize the energy consumption. The incorporation of DTVS and DVS produces significant improvement in energy saving regardless of the scheduling heuristic and workload. Simulation results affirm that our scheme produces better results than the current state of the art methodologies in terms of energy consumption. Moreover, the performance of the heuristics is also improved without violating targeted deadlines of the tasks.

In the recent years, numerous techniques have been presented to minimize the power consumption in clusters. Kriourov et al. [19] and Lang et al. [21] introduced slack in the execution time of jobs to achieve the aforementioned goal. Nevertheless, the proposed technique increases the overall execution time (makespan) of jobs giving rise to a trade-off between performance and energy consumption. The outcome of the above mentioned approach is undesirable especially in a real time job scheduling environment, where meeting the deadline is critical. A major research challenge is to focus on both the goals: (a) meeting deadline constraint and (b) minimizing the energy/power consumption of computing cluster.

This work hinges on the parametric model of estimated energy dissipation (EED) to find an appropriate DVS level. Nevertheless, the deadline constraint is not compromised that makes the scheduler efficient in terms of energy consumption. To circumvent the aforementioned problems, the PAJS perform the following three steps to accomplish energy minimization procedure (3EMP):

1. *Resource allocation* Identify the jobs (set of tasks) to be allocated to CNs and a set of task-to-node assignment is then formed.
2. *Resource matching* The algorithms dictate a CN-task pair that can meet the user defined deadline constraint.
3. *Resource scheduling* The algorithms determine the order of execution of tasks and the respective DTVS and DVS levels for each CN-task pair.

The key contribution of the work presented in this paper is the modeling and implementation of the PAJS, a task scheduler equipped with the DTVS and DVS modules. The notable contributions are apportioned as:

- We focus on minimizing the energy consumption using four defined DVS levels while maintaining the targeted deadlines.
- Eight heuristic based job scheduling algorithms are used to achieve energy efficient allocation of tasks to CNs.
- We have shown the adaptability of DTVS with the energy aware scheduling techniques. The results affirm that DTVS, when incorporated with DVS, further lowers the energy consumption as compared to the non-DTVS compliant version of the scheduling algorithms.

The analytical model proposed in this paper is implemented using MATLAB and analyzes eight probative task scheduling algorithms on the basis of: (a) makespan and (b) energy consumption. The PAJS methodology proposed here employs the set of eight heuristics based on greedy, recursive, and genetic algorithms. To maintain a fair comparison among the considered algorithms, the system parameters, such as the number of tasks and their respective deadlines, and execution time of tasks are kept similar. The eight heuristics performed are G-Min, G-Max, MinMax, G-Deadline, UtyFunc, ObjFunc, and two naturally motivated genetic heuristics, namely GenAlgo and GenAlgo-DVS. To investigate our simulations, variance in CNs and tasks heterogeneity is considered. To extend the scope of our analysis and evaluate the best solution on different data sets, the workload is varied from small sized workload (100 tasks) to large sized workload (100,000 tasks).

The remainder of the paper is arranged as follows. Related work is elaborated in Sect. 2. The problem formulation, system model, and implementation details of the PAJS are presented in Sect. 3. Discussion related to all of the heuristics is detailed in Sect. 4. Section 5 presents the simulation results and their comparison, while in Sect. 6 concluding remarks are presented.

2 Related work

The power management mechanisms in cluster computing can be categorized as: (a) dynamic power management (DPM), and (b) static power management (SPM) [30]. The SPM technique employs a flash storage and a pair of low power embedded CPUs to limit the peak power consumption. Lang et al. [20] exploited fast array of wimpy nodes

(FAWN) for achieving the benefits offered by the SPM. The FAWN methodology promises high efficiency when tested on nodes working at lower frequency. However, while solving non-parallelizable problems and working on indivisible data set size, the FAWN is inefficient [20].

In the recent years, significant attempts have been made to conserve energy consumption of clusters using the DPM. The technique proposed by Chaparro-Baqueero *et al*. [8] speculates the resource utilization and uses software along with power scalable modules to reduce the power consumption of the system [9,31]. For brevity, we focus on the DPM that can be categorized into: (a) Dynamic voltage scaling (DVS) [SiYuan13,KoK13], (b) dynamic frequency scaling (DFS), and (c) dynamic voltage and frequency scaling (DVFS) [28,31]. Kolodziej et al. [18] employed DVS scaling for intelligent power management. It is noteworthy that the DVS approach scales down the voltage level, V_{DD} according to the need of the node. As soon as V_{DD} is decreased, a net decrease in the energy consumption is observed. Though the aforementioned method yields promising results in soft real time tasks, the energy optimization may not be significant in hard real time job scheduling, where the completion time of tasks is of foremost importance [18]. However, our proposed scheme achieves energy efficiency without violating the targeted deadlines of the jobs.

In the DVFS approach, a node needs to be furnished with a DVFS unit to exploit the benefits of frequency and voltage scaling. A node can be categorized in either of the two modes: (a) active mode or (b) idle/sleep mode. The node with minimum activity is assigned the sleep mode, whereas the node with high activity is assigned the active mode. In case the node in active mode is overloaded, it is assigned more voltage lines and clock speed [29]. A recent work by Beloglazov et al. [7] and Kliazovich et al. [16] exploited sleep mode for attaining the energy saving benefit in data centers for cloud computing.

A similar approach is modeled in [18] and [CaV12] for energy optimization using the dynamic voltage and frequency scaling (DVFS). Although the above mentioned approaches attempt to enhance energy efficiency in clusters, the DVFS is inherently limited. For each node the minimum voltage level is defined, below which the nodes operate incorrectly. The DVFS is an energy efficient technique employed by network designers to reduce the energy dissipation [17]. The DVFS adjust the clock frequency in real-time based on the operational voltage of a processor [27]. The key idea is to provide the circuit just enough of speed and voltage that is required to process the assigned workload [10]. The DVFS technique performs a transition from a high-power state (of the processor) to a low-power state, when the workload reduces [29]. The aim of the DVFS is to reduce the dynamic power [6]. However, the PAJS employs the DTVS technique to attain power conservation irrespective of the mode (active/idle) of the node. Therefore, our proposed work minimizes both the static and dynamic power losses.

A related approach to the DVFS is DFS. The DFS needs to be incorporated within the system utilizing it, for example speed step by Intel and VIA Technologies product called as Long Haul [CaV12]. A dynamic change is made in frequency of the system and energy efficiency is achieved by scaling the clock speed dynamically. However, energy benefits of the DFS are limited, due to the fact that a reduction in the frequency reduces the speed of the system. Therefore, the overall performance is degraded. The majority of the modern day scientific work focuses on the DVS for intelligent power management [24].

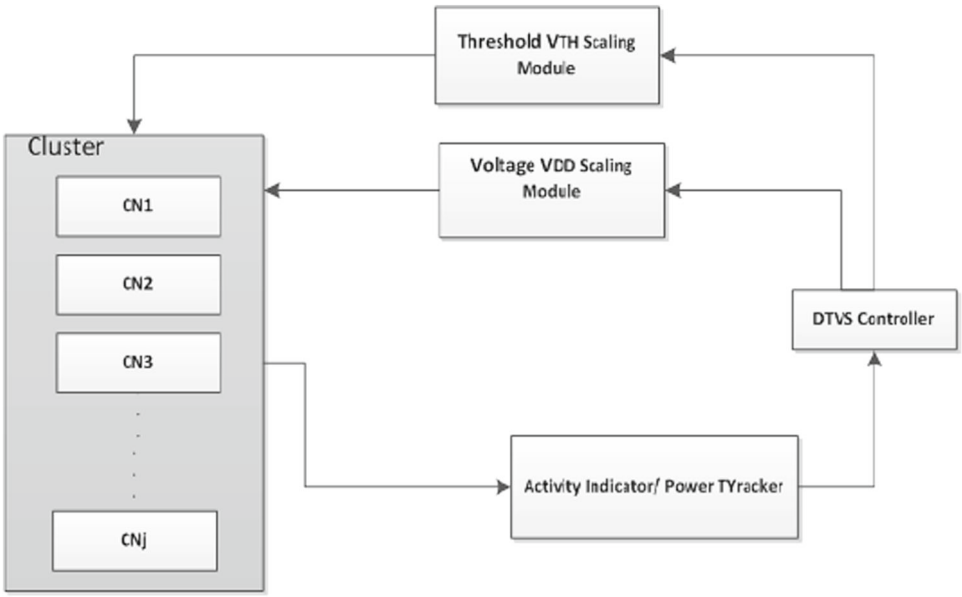
Alfonso et al. [3] proposed the cluster energy saving system (CLUES) tool, an energy management strategy for HPC clusters. The aforementioned methodology adjusts the number of working nodes by extracting information about the activity of nodes. A check on the inactivity time of the node is observed and if the check is successful, the particular node is turned off. Although the aforementioned approach reduces power consumption by minimizing the count of active nodes, it is prone to configuration issues. Moreover, such an approach might prolong the job wait time because of two reasons: (a) the delay incurred due to the check on inactivity of nodes and (b) addition of new nodes in the scheduler. Contrary to this approach, the benefit of the work presented in this paper is that it does not need to put a check on the activity level to maximize energy saving.

Valentini et al. [30] presents an extensive survey of the power management endeavors using the aforementioned methodologies. Surveys performed by Shuja et al. [27] and Usman et al. [29] presents a collection of interesting examples of the recently developed Power management schemes. Nevertheless, this line of work is sound in comparison to the existing state of the art as the PAJS combines the benefits of DVS with DTVS to meet the sine qua non for high performance and energy-efficient cluster.

3 Problem formulation

In this section, we discuss the basic parameters that affect the energy consumption of a cluster. For a CN two main modules are required to function: (a) voltage line and (b) clock speed, referred as frequency [22]. To address energy consumption issue, we want the system to be designed in a method that the above stated components of the framework are controlled judiciously. Both voltage and frequency share a linear relation (1), such as, decreasing the voltage decreases frequency. As a result, fewer computation cycles per unit time increase the execution time of the tasks. The time taken by the cluster to execute all of the assigned tasks is referred to as makespan in the paper.

Fig. 1 DTVS mechanism



$$f \alpha V. \tag{1}$$

The total power consumption P_t can be scripted as:

$$P_t = P_d + P_s, \tag{2}$$

where P_d is the dynamic power and P_s is the static power.

The dynamic power in Eq. (3) signifies that power is a quadratic function of voltage. This implies that a reduction in the supply voltage of the system will reduce the power consumption in a quadratic function

$$P_d = A \times V_{DD}^2 \times f_{CLK} \times C_{EFF}, \tag{3}$$

where, f_{CLK} is the clock frequency, V_{DD} is the supply voltage, A represents the activity factor, and C_{EFF} is the effective switched capacitance.

The energy consumption in a cluster is calculated using Eq. (4). To reduce the energy consumption, either the instantaneous power should be reduced or the computation time ought to be decreased. Therefore, power factor becomes a significant metric in the design of energy efficient clusters. Consequently, the focus should be on the power management.

$$Energy = Power \times Computation\,Time. \tag{4}$$

The DTVS procedure is illustrated with the help of a flow chart shown in Fig. 1. The motivation of using a power tracking mechanism is to monitor power consumption of each CN in the cluster. The power tracker/activity indicator module ascertains that the total power consumption level is within the tolerable bounds. When the total power consumed, P_{total} crosses its acceptable bounds, the DTVS module asserts the V_{TH} and V_{DD} scaling modules simultaneously, so as to regulate the total power P_t of the cluster. The power tracker/activity indicator also monitors if any CN in the cluster is idle. V_{TH} of the idle CNs is increased and V_{DD} is scaled down to a level that the CN is turned off, resulting in less power consumption. A list of acronyms and their meanings is summarized in Table 1.

3.1 The system model

We present a holistic model of a high performance cluster in this section. The aforementioned is comprised of a collection of CNs and works on a set of tasks, referred as a job.

Computing Nodes The set of CNs in the cluster is denoted as, CN = $\{CN_1, CN_2 \ldots CN_m\}$. It is assumed that each of the CNs is endowed with a DTVS as well as a DVS module. For the problem catered in this paper, we assume a constant and negligible transition time between successive levels of DVS as considered in [22].

Each CN is described by:

- The instantaneous power dissipation of the CN, C_j. The C_j may alter between C_j^{min} to C_j^{max}, depending on the DVS level of the CN. The range of C_j is thereof defined as $0 < C_j^{min} < C_j^{max}$.
- The specific computing node memory is abbreviated as m_{CNj}.

Tasks Consider a job, J, which is a collection of tasks. The set J= $\{t_1, t_2, \ldots\ldots\ldots\ldots, t_n\}$, where i_{th} task is represented by t_i. Each task in the job is characterized by:

- The deadline, $dead_i$ that needs to be fulfilled by the task.
- The computational cycles, C_i that a task needs to accomplish.
- The memory requirement of the task, m_{ti}.

Table 1 Notation/ acronyms and their meanings

Symbols	Meaning	Symbols	Meaning
DPM	Dynamic power management	G-Min	Greedy Heuristic that schedule shortest task
DVS	Dynamic voltage scaling		First
PAJS	Power-Aware Job Scheduler	G-Max	Greedy Heuristic that schedule longest task first
4EMP	4-Step energy minimization procedure	G-deadline	The tasks with the shortest deadlines are
t_{ij}	Run time of task t_i on CN_j		scheduled to the CNs first in this Greedy Heuristic
CN	Computing Node	MinMax	Greedy Heuristic that schedule tasks to the least
V_{DD}	CN supply Voltage		efficient CNs
M	Makespan	ObjFunc	Greedy Heuristic in which objective functions
J	Set of all t_i		are employed to govern task allocation
t_i	i^{th} task $\in J$	UtyFunc	Greedy Heuristic that schedule tasks based on a
CVB	Coefficient of variation based		utility function
$dead_i$	Deadline of t_i	GenAlgo	Genetic Algorithm
m_{ti}	Memory requirement of t_i	GenAlgo-DVS	Genetic Algorithm that utilizes DVS
m_{CNj}	Memory available to CN_j	C_i	Computational cycles required by t_i
DVS_k	k^{th} DVS level	I-ETE	Index for ETE Matrix
EED	Estimated energy dissipation	t_{ijk}	Run time of task t_i on CN_j at DVS_k
ETE	Estimated time of execution	m_j	Run time of computing node CN_j
CN	Set of CNs in CN allocation	$Energy_{idle}$	Energy consumed when CN is idle
CN_p	Set of CNs in CN pool	$Energy_j$	Energy consumed by CNj
CN_j	j^{th}CN $\in CN$	$DTVS$	Dynamic threshold and voltage scaling (DTVS)
p_{ij}	Power consumed by CN_j to execute t_i	μ_{task}	Average task execution time
E_{sol}	The best solution	V_{task}	Variance in execution time of tasks
C_j	Power consumption of CN_j	NIS	Naturally Inspired solutions
k_{idle}	Power scalar for idle CN	V_{CN}	Variance in computing node heterogeneity

Preliminaries We are given a set of CNs and a metaset of tasks, J. Each member of the set J has to be allocated to a CN such that the deadline constraint is not violated. A realistic task to node allocation is accomplished when:

1. The runtime m_j, of CN_j, is less than or equal to $dead_i$.
2. Each task, $t_i \in J$ is mapped to at least one CN_j if all the related constraints of each task are fulfilled.
3. For a successful mapping, $m_{CNj} > m_{ti}$ is satisfied. In case the aforementioned inequality is not satisfied, t_i cannot be assigned to CN_j.

3.2 Modeling the energy-makespan minimization problem

Given a metaset of tasks J, and a pool of CNs, the problem statement such that:

- The net power utilized by the CNs is minimized.
- A minimization in makespan, M, of the metaset of tasks, J is achieved.

The mathematical model of the problem statement can be expressed as:

$$min \sum_{i=1}^{n} C_{ij} x_{ij} \quad \text{such that} \quad minmax\, 1 \leq j \leq m \sum_{i=1}^{n} t_{ij} x_{ij}$$

Obliged to the subsequent constraints from 5 through 9:

$$x_{ij} \in \{0, 1\}, \quad where \quad i = 1, 2, \ldots\ldots\ldots m, \quad and \quad j = 1, 2, \ldots\ldots\ldots n, \tag{5}$$

$$If t_{i\rightarrow} m_j, \forall_i, \forall_j \quad such\, that \quad \mathrm{m_{CNj}} > mt_i, \quad then \quad x_{ij}, \tag{6}$$

$$(t_{ij} x_{ij} \leq dead_i) \in \{0, 1\}, \tag{7}$$

$$t_{ij} x_{ij} \leq \mathrm{dead}_i, \forall_i, \forall_j, x_{ij} = 1, \tag{8}$$

$$\prod_{i=1}^{n} (t_{ij} x_{ij} \leq \mathrm{dead}_i) = 1, \forall_i, \forall_j, x_{ij} = 1. \tag{9}$$

A task, t_i, from a job is assigned to CN_j when the mapping constraint (5) equals 1.This can be expressed as when $x_{ij} = 1$. Constraint (6) juxtaposes an additional constraint on the

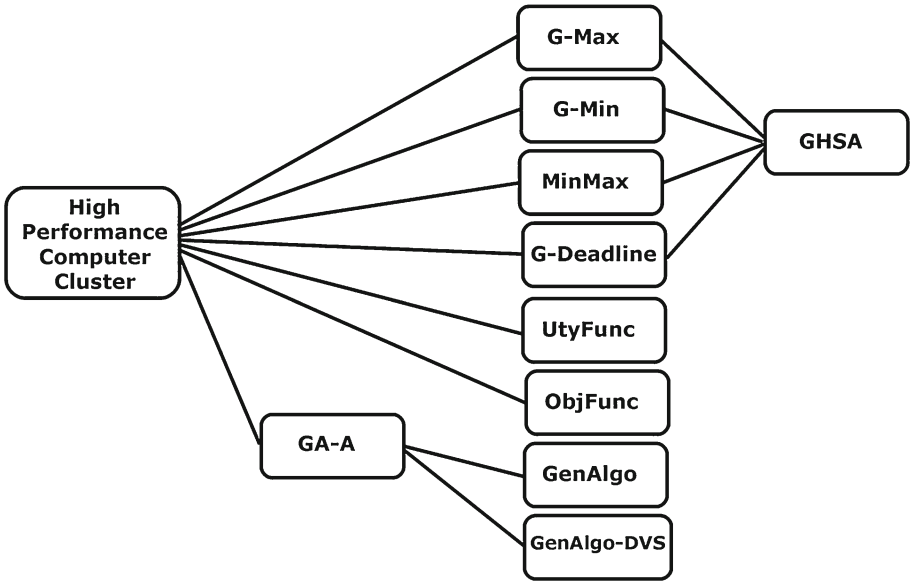


Fig. 2 Scheduling heuristics used by the PAJS

mapping procedure. The aforementioned mapping can take place only when the CN_j satisfy the memory requirement of *ti*. The Boolean relationship between the deadline of the task and its actual time of completion is depicted in constraint (7). Constraint (8) is adherence to the individual task deadline. The deadline constraint of the metaset is shown in (9) and is logically true if for each $t_i \in J$, the deadline, dead_i, is satisfied. The main aim of PAJS is energy minimization in computational clusters. In this paper, we discuss how to reduce makespan and minimize the system's overall power consumption at the same time. The aforementioned parameters make the PAJS problem formulation a multi objective and multi constrained optimization problem. The formulation is similar to [13] the power aware task allocation (PATA) and energy-aware task allocation (EATA), except for the additional constraints of the DTVS [22]. The major difference between the PAJS and the mentioned techniques is that the domain of PAJS is wider in terms of capacity of resources and computations performed.

4 Heuristics implementation and evaluation

This section presents the execution behavior of the eight algorithms analyzed and implemented in the implementation of the PAJS. Fig. 2 depicts the control of the eight scheduling heuristics employed for the analysis process. The estimated time of execution (ETE) matrix gives the task completion time on a specific node. The number of rows in the ETE depends on the tasks count in J, and the number of entries in CN_p indicates the column length of the ETE. Each entry (i,j) of the ETE matrix corresponds to the estimated execution time of task i on node j. In the ETE matrix, the entries in a row signify the estimated completion time of a particular task on different nodes whereas, elements along a column depict the estimated completion time of various tasks on a specified node. The Coefficient of Variance Based (CVB) method is employed for the generation of the ETE matrix [4]. The three parameters used to introduce heterogeneity in generation of the ETE matrix are:

- The variance in Computing Nodes heterogeneity, V_{CN}.
- The variance in execution time of tasks, V_{task}.
- The average completion time of each $t_i \in J$, μ_{task}.

The above listed heterogeneity parameters are incorporated in the generation of ETE matrix to imitate a workload that is supported in previous studies and is derived from real world applications [2,4,5,14,22]. The probability distribution function used by the CVB is a gamma distribution, so it is necessary to define the shape parameter, α, and scale parameter, β. The mean of the gamma distribution is $\mu = \beta\alpha$ and the variance is, $V = 1/\sqrt{\alpha}$. However, the parameters α_{task}, α_{CN}, β_{task}, and β_{CN} in the gamma distribution are interpreted as μ_{task}, V_{task}, and V_{CN} in this paper.

$$\alpha_{task} = 1/V_{task}^2, \tag{10}$$

$$\alpha_{CN} = 1/V_{CN}^2, \tag{11}$$

$$\beta_{task} = \mu_{task}/\alpha_{task}, \tag{12}$$

$$\beta_{CN} = G(\alpha_{task}, \beta_{task})/\alpha_{CN}, \tag{13}$$

where $G(\alpha_{task}, \beta_{task})$ is the gamma distribution's sample number.

Table 2 Power scales and operational speeds for different DVS levels

DVS level	Power scale	Operational speed (%)
1	0.3430	70
2	0.5120	80
3	0.7290	90
4	1.0000	100

The deadline, $dead_i$ for each task within a job $t_i \in J$ is derived from the ETE matrix and is given as

$$\text{dead}_\text{i} = \frac{|t_i|}{|CN|} \times arg_j \max(\text{ETE(i, j)}) \times k_d, \tag{14}$$

where parameter k_d is used to tighten the deadline $dead_i$ [22].

4.1 Greedy heuristics

4.1.1 Greedy Heuristic Scheduling Algorithm (GHSA)

The GHSA accomplishes resource scheduling for the algorithms discussed in the subsequent sections, i.e. Section 4.1.2 to 4.1.5. The methodology used by each of the above-mentioned heuristics for scheduling J to CN varies. The pseudo-code of GHSA is presented in Algorithm 1. The inputs of the GHSA encompass $ETE, CNp, dead_i \forall t_i \in J$, and CN. GHSA produces a J to CN mapping, M, and E_{sol} subject to the inputs defined.

The GHSA initiates the mapping procedure by assigning the first entry (task) of the ETE matrix to the best suitable CN. To ensure a feasible mapping the dead_i constraint, stated in line 4 should be satisfied. The mapping of t_i to CN_j is to adhere to the minimum possible level of DVS scaling, DVS_K (Table 2). Starting from DVS_1, the DVS_K is increased when $dead_i$ is not satisfied. The DVS_K at which dead_i constraint agrees, no further increments are applied to DVS_K. In case the dead_i is not met by the task even at the extreme DVS level (DVS_4), then the particular t_i is allocated to the next CN in the ETE matrix by GHSA. Line 10 illustrates that if the deadline is not compiled by GHSA in scheduling t_i on any of the CNs, then a flag, d_{flag} is activated. The d_{flag} indicates the absence of a feasible solution for a specific t_i. After the successful assignment of t_i to a CN, both the execution time of t_i and the energy consumption of CN_j are recorded. Line 6 corresponds to the addition of ETE (i,j) to m_j.

Line 7 signifies the addition of EED (i,j) to E_{sol}. For any feasible solution obtained, t_i to CN mapping is processed and the energy consumed is computed. The idle time energy in Line 16 is calculated by Eq. (15) that is:

$$Energy_{idle} = CN_j \times time_{idle} \times k_{idle}. \tag{15}$$

Where k_{idle} is the DVS_K level of an idle CN and time_{idle} is given by the equation (16) that is:

$$time_{idle} = M - m_j. \tag{16}$$

Algorithm 1: Greedy Heuristic Scheduling Algorithm (GHSA)

Input: $ETE, CN_p, dead_i \forall t_i \in j, CN$

Output: $J\ to\ CN\ mapping, E_{sol}, M$

```
1:  foreach t_i ∈ j do
2:      CN_j ∈ CN do
3:      for DVS_k = 1 to 4 do
4:          if t_ijk + m_j ≤ dead_i then
5:              Assign t_i to CN_j at DVS_k;
6:              m_j ← m_j + ETE(ij);
7:              E_sol ← E_sol + EED(ij);
8:          end
9:      end
10:     if t_i not assigned then
11:         d_flag ← 1;
12:         EXIT;
13:     end
14: end
15: foreach CN_j ∈ CN do
16:     E_sol ← E_sol + Energy_idle;
17: end
18: foreach t_ijk
19:     if t_ijk > m_j and t_ijk < M
20:         DTVS = 0;
21:     else
22:         DTVS = 1;
23: end
```

From line 18 to 23, the run time t_{ijk}, for each task $\text{t}_i \in$ J on a particular CN at a specific DVS_k level, is compared with the run time, m_j, of that CN and the overall makespan, M. If t_{ijk} is greater than m_j but less than M, this indicates that the CN_j is idle. To conserve energy at this idle time of the CN_j, the DTVS mechanism is employed to the node.

4.1.2 G-Min

The name of the algorithm 2 suggests that the tasks are greedily scheduled based on the shortest task first and is named as G-Min. The goal of executing the tasks in aforementioned

manner is to introduce a slack in scheduling of the tasks. The main benefit of using a slack is that it allows the scheduler to schedule the subsequent tasks with longer run-times without violating their individual deadlines. Input parameters of G-Min comprise of an ETE matrix. The output is thereof a rearranged ETE matrix, an EED matrix, and I-ETE. In Algorithm 2, R signifies a row in ETE matrix, whereas, C_i stands for the i^{th} column in ETE matrix.

It is interesting to note that MinMax G-Deadline, G-Min, and G-Max share identical input and output parameters. Once the elements of the ETE matrix are rearranged, the GHSA is applied to the ETE matrix for evaluation purposes. The minimum power consumption is achieved by computing ETE for different DVS levels using the DVS methodology.

4.1.3 G-Max

The greedy heuristic that prefers to execute the longest task first is termed as G-Max and the scheduling mechanism is depicted in Algorithm 3. Therefore, the leftover tasks to be scheduled are the ones with the shorter execution times. First, each row is rearranged in ascending order. Then the rows are swapped in such a manner that the first column of the resulting ETE matrix is arranged in descending order. The crux behind this scheduling scheme is that the tasks with shorter execution times are more easily scheduled by GHSA without contravening the deadline constraint.

Algorithm 2: G-Min

Input: *ETE*

Output: *ETE, EED, I-ETE*

1: **foreach** row, $R \in ETE$ **do**
2: | Sort R and corresponding row in I-ETE in ascending order;
3: **end**
4: $\forall\ R\ \in ETE$, interchange *R's* such that C_1 is in ascending order;
5: Make similar changes in *I-ETE* with respect to step 4;
6: INVOKE GHSA;

Algorithm 3: G-Max

Input: *ETE*

Output: *ETE, EED, I-ETE*

1: **foreach** row, $R \in ETE$ **do**
2: | Sort R and corresponding row in I-ETE in ascending order;
3: **end**
4: $\forall\ R\ \in ETE$, interchange *R's* such that C_1 is in descending order;
5: Make similar changes in *I-ETE* with respect to step 4;
6: INVOKE GHSA;

4.1.4 G-Deadline

In algorithm 4 the tasks are scheduled based on the criterion of their respective deadlines and is named as G-Deadline. The tasks scheduled first are the ones with the shortest deadlines. The task with less sensitive deadline constraint can be lingered on and scheduled later in the scheduling order. The working of the algorithm demands a two-step reordering procedure of the ETE matrix: (a) Initially, the ETE matrix is arranged such that the individual rows are in ascending order and (b) then a swapping of rows is done such that the elements in the first column are assembled in ascending order based on the task's deadline. The GHSA is invoked once the ETE matrix is re-arranged, by the G-deadline scheduling routine.

4.1.5 MinMax

Scheduling of tasks in algorithm 5 is achieved on the basis of efficiency of CNs and is named as MinMax. The initial phase of MinMax allocated the task to the least efficient CNs. The main goal is to introduce a slack in the scheduling process. The next phase schedules the subsequent tasks on the most efficient CNs. Firstly, the rows in the ETE matrix are ordered in descending order. Finally, the swapping of rows is performed such that the first column is aligned in descending order based on task completion times.

4.1.6 ObjFunc

To ascertain a power efficient task to CN mapping the greedy heuristic objective function is employed, abbreviated as ObjFunc in the paper. This heuristic utilizes the following two objective functions for the above stated purpose: (a) task selection and (b) node selection. Algorithm 6 depicts the pseudo-code for ObjFunc. The inputs to ObjFunc comprises of $ETE, CNp, dead_i \forall t_i \in J$, and CN. The output renders $JtoCN$ mapping, M and the most optimized energy solution, E_{sol}.

The algorithm starts with the selection of a task for which a select task array (ST) is generated. The ST array has an entry for every t_i that is to be assigned to one of the CN for the above said purpose. The following objective function is used for selection:

$$ST = \alpha_1(T_{2,i} - T_{1,i}) + \alpha_2(N_{2,k} - N_{1,k}) + \alpha_3 \frac{T_{1,i} + T_{2,i}}{\sum_{j=1}(T_{1,j} + T_{2,j})} + \alpha_4 + \alpha_5 + \alpha_6, \quad (17)$$

where $T_{1,i}$ and $T_{2,i}$ represent the minimal and second minimal estimated execution time of task, t_i, respectively. $N_{1,k}$ is the most power-efficient and $N_{2,k}$ is the second most power-efficient CN for the execution of task t_i, respectively. The components α_1,α_2, and α_3 are the weight parameters and

components α_4,α_5, and $\alpha_6$are scalars added to ST array in case the following conditions are satisfied:

- α_4 is added to the ST array if the most power-efficient CN is also the one with shortest t_{ij}.
- α_5 is added to the ST array if the second most power-efficient CN is also the one with shortest t_{ij}, or *vice-versa*.
- The scalar α_6 is added to the ST array if the second most power-efficient CN and the CN with second shortest t_{ij} are same.

Line 4 depicts the sorting of ST in descending order to ensure that ObjFunc schedules the most appropriate task first. In order to determine the most suitable CN for each task, Line 7 is coded and the result is recorded in CN select array, SN.

Algorithm 4: G-Deadline

Input: *ETE*

Output: ETE, EED, I-ETE

1: **foreach** row, $R \in ETE$ **do**
2: Sort R and corresponding row in I-ETE in ascending order according to each t_i's $dead_i$;
3: **end**
4: $\forall R \in ETE$, interchange R's such that C_1 is in ascending order on the basis of vector $dead_i$;
5: Make similar changes in *I-ETE* with respect to step 4 ;
6: INVOKE GHSA;

Algorithm 5: MinMax

Input: *ETD*

Output: ETD, EED, I-ETE

1: **foreach** row, $R \in ETE$ **do**
2: Sort R and corresponding row in I-ETE in ascending order;
3: **end**
4: $\forall R \in ETE$, interchange R's such that C_1 is in ascending order;
5: Make similar changes in *I-ETE* with respect to step 4;
6: INVOKE GHSA;

Algorithm 6: ObjFunc

Input: $ETE, CN_p, dead_i \; \forall \; t_i \in J$, and CN

Output: J *to* CN mapping E_{sol}, M

1: **foreach** $ti \in J$ **do**
2: Calculate ST_i
3: **end**
4: Sort ST in descending order;
5: **foreach** $ti \in ST$ **do**
6: **foreach** $CNj \in CN$ **do**
7: Calculate $SNij$;
8: **end**
9: $j \leftarrow arg_j \, min(SNij)$
10: **for** $DVS_k = 1$ *to* 4 **do**;
11: **if** $t_{ijk} + m_j \leq dead_i$ **then**
12: Assign t_i*to* CN_j *at* DVS_k;
13: $m_j \leftarrow m_j + ETE(ij)$;
14: $E_{sol} \leftarrow E_{sol} + EED(ij)$;
15: **end**
16: **end**
17: **if** ti not assigned **then**
18: $d_{flag} \leftarrow 1$;
19: **EXIT**
20: **end**
21: **end**
22: **foreach** $CNj \in CN$**do**
23: $E_j \leftarrow E_j + E_{idle}$;
24: **end**
25: **foreach** t_{ijk}
26: **if** $t_{ijk} > m_j$ and $t_{ijk} < M$
27: DTVS = 0;
28: **else**
29: DTVS = 1;
30: **end**
31: **end**

For every task the selection function assigns a value to each CN using the objective function given by the following equation.

$$SN = \beta_1 T_{1,CN_{k,ti}} + \beta_2 N_{1,CN_{k,ti}} + \beta_3 load(CN_k), \qquad (18)$$

where $T_{1,CN_{k,ti}}$ corresponds to the completion time of task t_i on node CN_k, $N_{1,CN_{k,ti}}$ corresponds to the instantaneous power consumption when task t_i is executed on node CN_k and *load* (CN_k) is a value dependent on certain conditions. If t_i satisfies $dead_i$, the value of load (CN_k) equals zero otherwise, it is equal to m_j-$dead_i$. The intention behind this is to assign t_i to the CN having the lowest SN value. The lowest level DVS_k that CN_j can be assigned is determined by ObjFunc at line 12. Once t_i is scheduled on CN_j, the energy consumption of CN_j and the completion time of t_i must be recoded. Next, line 13-14 adds *ETE (ij)* to m_j and *EED (ij)* to E_{sol}, respectively. If the condition at Line 10 is not satisfied even when CN_j runs at the highest DVS_k, then this indicates that a workable solution does not exit, and a flag is set.

The net energy consumption of the solution is evaluated in case a feasible solution is achieved. The values of parameters $\alpha_1$to α_6 and β_1 to β_3 are presented in Table 3. In lines 25 to

Table 3 Parameters used in task select array and cn select array

Parameters	Value
β_1	0.0970764
β_2	0.4008180
β_3	0.7734070
α_1	0.5206560
α_2	0.3819580
α_3	0.0431519
α_4	0.1605830
α_5	0.5223390
α_6	0.6965640

30, DTVS is employed to ensure power aware scheduling. The status of each CN_k is evaluated, if the check result in an idle status for the CN_k, then DTVS is set to zero. Otherwise the value of DTVS is set to one.

4.1.7 UtyFunc

For the task to CN assignment the heuristic utilizes a utility function and is abbreviated as UtyFunc. The purpose of using the utility function is that it determines the advantage obtained from each task to CN allocation. Algorithm 7 depicts the pseudo-code of UtyFunc. The inputs to UtyFunc are dead$_i \forall$ t$_i$ $\in$J, ETE matrix, CN, and instantaneous power for each node. The output of UtyFunc is the J to CN mapping, Energy$_{sol}$, and makespan of the most energy efficient node. In Line 4, the UtyFunc iterates to compute the utility of every task for each of the CN and the corresponding DVS level. To correlate speed and execution time, the utility function is employed. The speed and utility can be depicted as:

$$\text{Speed(V)} = \frac{K_1(V - V_t)^2}{V}, \tag{19}$$

$$\text{Utility} = (\text{Speed})^p \times (\text{Time})^q, \ \ (\text{Such that p and q } > 0) \tag{20}$$

In Eq. 20, p ascertains the significance of the speed whereas q represents the relative importance of the completion time of the task. Based on the highest utility yielded by the CN and DVS level, the process of task assignment is concluded. However, it is noteworthy to mention that the assignment of t_i to the highest utility CN_j does not ensure the fulfillment of the deadline constraint. As depicted in Line 15, the utility of the particular t_i-CN_K pair is set to zero if the deadline is violated. The UtyFunc in such a scenario recognizes the CN–DVS level pair for the particular task that guarantees the next highest utility and allocates the task to the identified CN–DVS level pair.

Algorithm 7: UtyFunc

Input: $ETE, CNp, dead_i \ \forall \ t_i \in J$, and CN

Output: J *to* CN, E_{sol}, M

```
1:  foreach ti ∈ J do
2:      foreach CNj ∈ CN do
3:          for DVSk=1 to 4 do
4:              Calculate Utijk;
5:          end
6:      end
7:  end
8:  foreach ti ∈ J do
9:      j,k ← arg j,k max(Utijk);
10:     if tijk + mj ≤ deadi then
11:         Assign ti to CNj at DVSk;
12:         mj ← mj + ETE(ij);
13:         Esol ← Esol + EED(ij);
14:     else
15:         Utijk ← 0;
16:     end
17:     if ti not assigned then
18:         dflag ← 1
19:         EXIT
20:     end
21: end
22: foreach CNj ∈ CN do
23:     Esol ← Esol + Eidle;
24: end
25: foreach tijk
26:     if tijk > mj and tijk < M
27:         DTVS = 0;
28:     else
29:         DTVS = 1;
30:     end
31: end
```

For a successful task allocation the power consumption of CN$_j$ and execution time of t$_i$ are added to E$_{sol}$ and m_j, respectively. A d$_{flag}$ is set when a task is unable to be assigned to any of the CNs even though the deadline is satisfied. The main point behind enabling the d$_{flag}$ is to indicate an absence of a feasible solution. In case an acceptable solution is found the UtyFunc algorithm, then determines the net energy consumption in a similar manner as GHSA and ObjFunc. The application of DTVS is in the same way as discussed for GHSA and ObjFunc.

4.2 Genetic algorithms

A class of evolutionary algorithm that employs searching and optimization techniques is known as Genetic algorithms. The

main objective is to find the most suitable solution among a randomly generated population. In order to achieve the above stated objective, we first randomly generate an initial population of solutions using the genetic algorithm. The genetic algorithm performs the following four steps recursively until it encounters a halting condition.

1. *Solution Ranking* The solutions ranking is based on their Distance from Origin (DFO).
2. *Solution Grading* The solutions with the same DFO are then checked for their Distance from Line (DFL).
3. *Reproduction* The solutions with the highest ranking are improved through *mutation* and *crossover*.
4. *Substitution* The solutions with lowest rank are discarded and substituted by the newly produced solutions.

The traversal and completion of the above stated steps, completes a generation. The halting condition is verified soon after the appraisal step. In genetic algorithm, each solution produced by the algorithm is characterized by a chromosome. Figure 3 represents a chromosome. The chromosome depicts information about each task and its mapping on a specific *CN*. A *CN* may be assigned more than one task. The scheduling order of the task recalls the Step 3 of EMP and is attained by invoking GenAlgo or GenAlgo-DVS. Both of these are explained in the subsequent text.

The *PAJS* framework utilizes distance from origin (DFO) approach to rank the solutions produced. One particular solution dominates the other solution if its DFO value is less as compared to others. The PAJS framework is multi-Objective and examines two metrics: (a) makespan, and (b) energy consumption.

A generalized multi-objective optimization function can expressed as follows:

$$minF(y) = (f_1(y), f_2(y), \ldots\ldots\ldots, f_n(y))^T | y \in S$$

$$y = (y_1, y_2, \ldots\ldots\ldots, y_n)^T, \tag{21}$$

where $(y_1, y_2, \ldots\ldots\ldots, y_n)$ are the optimization parameters, $f_1(y), f_2(y), \ldots, f_n(y)$ are the objective functions, and S is the parameter or solution space. The solution space S is mapped onto Y, by F. Figure 3 shows the dominated solutions. The solutions with the same DFO are compared by taking DFL, which is a 45o angle line, to ensure an equal importance of each metric. Moreover, PAJS is also capable of producing diversity in its objectives weights by modifying the slope of the straight line employed to determine DFL.

The solutions are ranked based on the Pareto optimality that uses non-dominated sorting for the ranking purpose [12]. First, the non-dominated solutions are figured out. These solutions are given the highest rank i.e. they are represented by one and removed from the population. Now, in the reduced set of population the same procedure is repeated to identify the non-dominated solutions. The result obtained by doing so

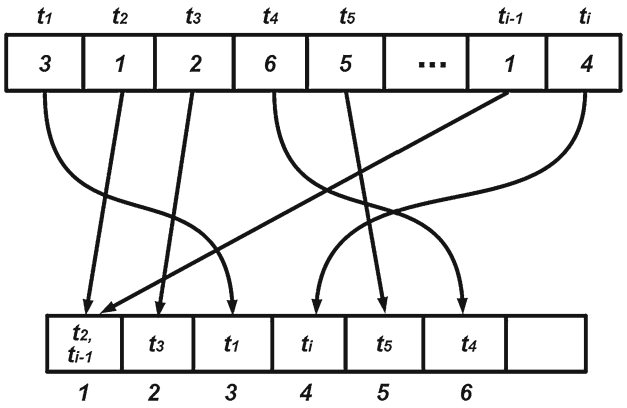


Fig. 3 Chromosome representation

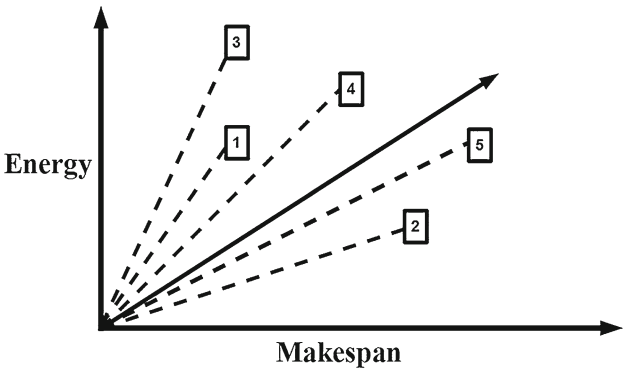


Fig. 4 Representing Pareto ranking in the PAJS framework

is given a rank two, and the solutions with this particular rank are removed from the population pool. This process is repetitive and it goes on until the entire population of solutions is ranked as shown in Fig. 4.

To maintain genetic diversity the genetic algorithm employs two genetic operators: (a) Crossover and (b) Mutation. The genetic operator, crossover, is a used to exchange information between two parent chromosomes. GA-Algo performs a two cut-point crossover technique. The same points are selected in both the parent chromosomes to be the cut points. Swapping of information is performed amongst the two cut points to create child chromosomes as illustrated in Fig. 5. It is evident from the child chromosomes that the swapping of tasks has been performed between the two cut points in the parent chromosome. Mutation is a genetic operator that ensures diversity in solution along the generations. In order to figure out the best solution, the tasks in a solution are reassigned randomly to a different CN. This strategy incorporates heterogeneity and explores the most energy saving mapping of a task onto a CN.

4.2.1 GA-Algo

To maintain diversity in the generated population, algorithm 8 is employed and is termed as GA-Algo. In order to pro-

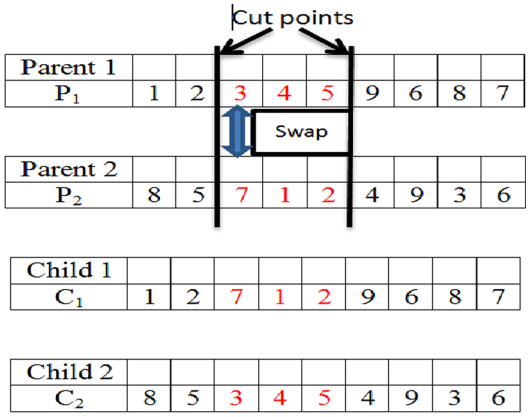


Fig. 5 Crossover

vide effective guidance in choosing the best solution cluster of CNs are created. Algorithm 8 illustrates the procedure of GA-Algo.

- *Initializations*
  - GA-A algorithm comprises of inputs such as ETE matrix, $dead_i \forall t_i \in J$, CN_p, and instantaneous power.
  - Initial solutions are randomly generated by GA-A (Line2) and then GenAlgo or GenAlgo-DVS is invoked for the evaluation of the initial solutions generated.

Algorithm 8: GA-Algo

Input: ETE, CNp, $dead_i \forall\, t_i \in J$

Output: J *to* CN *mapping*, $\mathcal{E}_{min}$, M

```
1:  Generate Clus;
2:  Initial Solutions Generation;
3:  INVOKE GenAlgo / GenAlgo-DVS;
4:  while k1 ≤ k1,max do
5:      while k2 ≤ k2,max do
6:          Ranking of Solutions;
7:          Solution Reproduction carried out;
8:          INVOKE GenAlgo/GenAlgo-DVS;
9:          if any Sn ∈ Clusm Improved then
10:             k2 ← 0;
11:         else
12:             INCREMENT k2
13:         end
14:     end
15:     if Any Clusm ∈ Clus Improved then
16:         k1 ← 0;
17:     else
18:         INCREMENT k1;
19:     end
20:     Rank Clusters;
21:     Cluster Reproduction carried out;
22:     INVOKE GenAlgo/GenAlgo-DVS;
23: end
```

Solution evaluation (Line 3–5)

Either of the GenAlgo or GenAlgo-DVS is invoked, the initial solutions are evaluated until the halting condition, and the two energy minima's are calculated: (a) the local energy minima, E_{min}, and (b) the global energy minima, γ_{min} for every generation.

- *Aggregating diversity (Line 10–14)*

The solutions with the most dominating rank are chosen for reproduction using genetic operator's crossover or mutation. However, in each cluster $Clus_m$, the solutions with lowest rank are discarded and replaced by the newly reproduced high-ranked solutions. As previously described GA–A involves GenAlgo or GenAlgo-DVS for the evaluation of the solution. In case a solution with energy efficiency better than the previous solutions is found, zero is assigned to k_2 to terminate the inner while loop and $Energy_{min}$ is updated. Otherwise the value of k_2 is incremented.

- *Cluster ranking (Lines 15–23)*

In any of the cluster if the local energy minima become less than the global minima, k_1 is assigned zero. Otherwise the value k_1 is incremented. Cluster ranking based on the solution quality is performed. Considering the cost of nodes making the clusters, cluster domination, $Clus_{dom}$ is used to rank the clusters. Mathematically, C_{dom} can be expressed as:

$$Clus_{dom}(a,b) = \max(x \in NIS(x)) \sum_{y \in NIS(b)} DOM(x,y), \tag{22}$$

$$Clus_{rank}[a] = \sum_{b \in C, \forall a \neq b} Clus_{dom}(a,b). \tag{23}$$

The above expressions reveal that the sum of $Clus_{dom}$ value for each cluster can be used to rank the clusters. Once the clusters are ranked, the reproduction of cluster is performed in a similar manner to reproduction of solutions, to establish diversity in out experimental setup, the reproduced clusters are improved using mutation and crossover to report the best chromosome as the final solution.

4.2.2 GenAlgo

To cater the task scheduling and evaluation of solutions produced by GA–Algo (step 3–4 of EMP) GenAlgo is employed. Algorithm 9 depicts the pseudo-code for GenAlgo. The input consists of an *ETE matrix*, $dead_i \forall t_i \in J$, *Clus*, CN_p and *CN*. The output are E_{S_n} and M.

- *Solution evaluation (Lines 1–3)*

Before scheduling the task on the nodes available in each cluster *Clus*, every solution in *Clus* is evaluated iteratively.

Because the task scheduling is not carried out yet, so the energy consumption of Solu_n is set as zero.

- *Task scheduling (Line 4–8)*

Iterations are performed through Solu_n to figure out the most appropriate CN. It is observable that feasibility of CN is satisfied only if the deadline constraint is not violated. When a successful allocation takes place, the energy dissipation and runtime of t_i is added to $CN_j's$ total energy consumption E_{S_n} and runtime (m_j) respectively.

- *Invalid solutions (Line 9–10)*

Taking into consideration the deadline constraint, a solution is termed as invalid if the aforementioned condition is not satisfied.

- *DTVS mechanism (Line 15–20)*

It can be gathered from the above description of GenAlgo that it does not consider CN's DVS level. The DTVS mechanism is employed to conserve the energy of the nodes that are in idle mode.

Algorithm 9: GenAlgo

Input: $ETE, EED,\ deadi\ \forall\ ti\ \in J, Clus,$ and J

Output: $E_{S_n}\ \forall\ S_n \in Clus$ and M

```
1:  foreach Clus_m ∈ Clus do
2:      foreach Solu_N ∈ C_m do
3:          Energy_{S_n} ← 0;
4:          foreach t_i ∈ J do
5:              if ETE(ij) + m_j ≤ dead_i then
6:                  Assign t_i to CN_j;
7:                  m_j ← m_j + ETE(ij)
8:                  E_{S_n} ← E_{S_n} + EED(ij);
9:              else
10:                 Solu_n is invalid;
11:             end
12:         end
13:     end
14: end
15: foreach t_ijk
16:     if t_ijk > m_j and t_ijk < M
17:         DTVS = 0;
18:     else
19:         DTVS = 1;
20:     end
21: end
```

Algorithm 10: GenAlgo-DVS

Input: $ETE, EED,\ di\ \forall ti\ \in T, C,$ and T

Output: $E_{sol}\ \forall\ S_n \in Clus$ and M

```
1:  foreach Clus_m ∈ Clus do
2:      foreach S_N ∈ Clus_m do
3:          E_{S_n} ← 0;
4:          foreach t_i ∈ T do
5:              t_a ← 0;
6:              k ← 1;
7:              while t_a = 0 & k ≤ 4 do
8:                  if t_ijk + m_j ≤ d_i then
9:                      Assign t_i to CN_j at DVS_k;
10:                     t_a ← 1;
11:                     m_j ← m_j + t_ijk
12:                     E_{S_n} ← E_{S_n} + EED(ij);
13:                 else
14:                     k ← k + 1;
15:                 end
16:             end
17:             if t_a = 0 then
18:                 Solu_n is invalid;
19:             end
20:         end
21:     end
22: end
23: foreach t_ijk
24:     if t_ijk > m_j and t_ijk < M
25:         DTVS = 0;
26:     else
27:         DTVS = 1;
28:     end
29: end
```

4.2.3 GenAlgo-DVS

In Algorithm 10 the CN's DVS module is exploited and is termed as GenAlgo-DVS due to the fact that DVS levels are used in the algorithm. The technique used for task evaluation and scheduling is similar to GenAlgo, with the exception that the mode power level is now governed by its DVS level. The main intuition behind GenAlgo-DVS is that it will enable the programmer to analyze the energy consumption (using DVS module) and compare it to GenAlgo. Though the input and output parameters of both GenAlgo and GenAlgo-DVS are the same, except that the later uses two additional variables. The two variables introduced are t_a and k. The variable k determines the DVS level and t_a determines whether the task has been allocated or not. If a task cannot be assigned at any of the DVS level due to the deadline constraint, then it is

said to be as invalid. A zero value of t_a signifies that the task has not been scheduled to any CN, however a 1 determines a successful assignment.

From line 23 to 28, the run time t_{ijk}, for each task $t_i \in J$ on a particular CN at a specific DVS_k level, is compared with the run time, m_j, of that CN and the overall makespan, M. If t_{ijk} is greater than m_j but less than M, this indicates that the CN_j is idle. To conserve energy at this idle time of the CN_j DTVS mechanism is employed on the node

5 Simulation results and discussion

The MATLAB is very efficient at performing operations and solving large sized matrices [23]. Therefore, the heuristics discussed in this paper were implemented using MATLAB. The ETE matrices used in our simulations had dimensions as large as 100,000 tasks by 20 Machines. The simulations were performed on 3.4 GHz Core i7 processor with 8 GB of RAM.

The objectives for our simulation study are

- To measure the Energy consumption and Makespan for different sets of tasks on a pool of machines using the eight scheduling heuristics.
- To measure and compare the results of energy consumption of the discussed eight heuristics while employing DTVS and without employing DTVS.
- To study the effect of change in system parameters on the performance of all heuristics. The summary of system parameters is presented in Table 4.

5.1 Workload

The simulation experiments are divided in three categories on the basis of workload (the total number of tasks) namely small sized, medium sized, and large sized workloads. The simulations that were executed for 100 and 1,000 tasks were placed in small sized workloads. The simulations executed for 10,000 tasks were placed in medium sized workloads, and for large sized workloads 100,000 tasks were considered in the simulations. The GenAlgo and GenAlgo-DVS were not computed for the large and medium sized workloads because of their large execution times [22].

The workload ETE matrix was generated using CVB ETE generation method as discussed in Sect. 4. The variance in tasks, V_{task}, and variance in CNs, V_{CN}, ranged between 0.1 and 0.35 while a value of 10 was set for μ_{task}, the mean task execution time as in previous studies [22]. To incorporate the variance in the generation of ETE workload, the parametric values are chosen from real world applications as mentioned in [2,4,5,14,22]. As discussed in Sect. 3, the deadline scaling parameter k_d is used to induce heterogeneity in deadlines $dead_i$, and is ranged from 1 to 1.8 [22]. The number of computing nodes, CNs, was set to 20 for all workloads in our simulations [4]. The number of DTVS and DVS levels was chosen to be 2 and 4, respectively [22]. The number of DTVS levels can be increased but having 2 DTVS levels serves better due to the fact that the CNs are either busy in executing some task or they are idle. The reduction in energy consumption is the main goal when the Computing nodes (CNs) are idle. In the active mode the major energy savings are achieved by using various DVS levels.

5.2 Results and discussion

The simulations were carried out on small, medium, and large sized workloads by varying three parameters namely, V_{CN}, V_{task}, and k_d. The aforementioned parameters are varied for our simulations and their results are compared. The simulations were carried out fifteen times for each set of parameters (27 sets of parameters are obtained by varying V_{CN}, V_{task}, and k_d), making a total of 405 simulations for all of the workloads and parameters. Table 5 summarizes the workloads and their respective set of parameters used for simulations.

The graphs of the simulation results give a great deal of information. The bold black horizontal line in all of the following graphs represents the grand mean of all of the heuristics. The black box represents the mean of the individual heuristic. The ± 1 and ± 1.5 times the standard deviation is represented by the gray box and black whiskers, respectively. Whereas the hexagons represent the outliers and the asterisk is used to represent extremes.

5.2.1 Small sized workloads

100 tasks: We started the evaluation of the heuristics with the small sized workload. Figure 6a shows the makespan comparison of all of the heuristics for 100 tasks. GenAlgo performed best for this workload while MinMax and UtyFunc were amongst the worse, when comparing makespan. The GenAlgo depicts the least makespan than any of the other heuristics, because GenAlgo does not employ DVS methodology. The G-Min, G-Max, and G-Deadline employ DVS scheme and exhibit almost similar results in terms of makespan. In terms of energy consumption without using the DTVS scheme on this workload, the comparison is depicted in Fig. 6b. The G-Max, G-Deadline, and G-Min utilized the least amount of energy and produced similar results when scheduling 100 tasks. ObjFunc produced results with minimum standard deviation and having a lower amount of maximum and minimum energy consumption. MinMax, ObjFunc, UtyFunc, and GenAlgo all had average energy consumption that was more than the grand mean. It can be observed that all the heuristics produced results with minimum outliers due to the small sized workload.

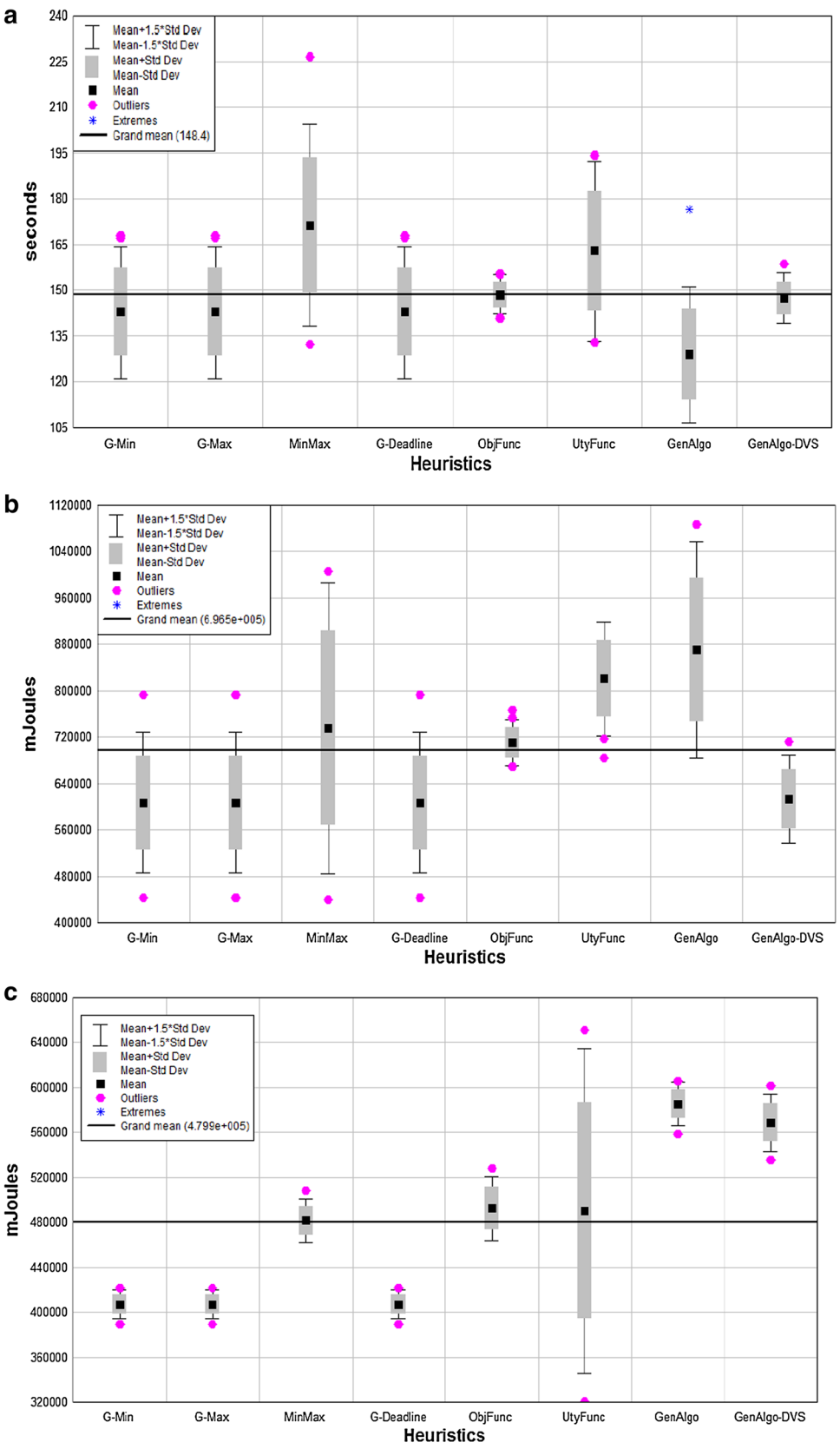


Fig. 6 Makespan for 100 tasks (V_{task}= 0.1, V_{CN}=0.1, k_d= 1) Energy Consumption (without DTVS) for 100 tasks (V_{task}= 0.1, V_{CN}= 0.1, k_d= 1) Energy Consumption (with DTVS) for 100 tasks (V_{task} = 0.1, V_{CN}= 0.1, k_d = 1)

Table 4 Summary of system parameters

System parameters		
Deadline scaling variable	k_d	(1, 1.3, 1.8)
Variance in task execution time	V_{task}	(0.1, 0.15, 0.35)
Variance in CN heterogeneity	V_{CN}	(0.1, 0.15, 0.35)
Number of tasks (workloads)	|T|	(100; 1,000; 10,000; 100,000)
Average execution time	μ_{task}	10
Number of Computing nodes	|CN|	20
Number of DVS levels employed	DVS_k	4
Number of DTVS levels employed	$DTVS_k$	2

Table 5 Summary of workloads and parameters considered for simulations

No.	Category	No. of tasks, No. of CNs	System parameters
1.	Small sized workloads	100 tasks, 20 CNs	V_{CN} = 0.1, V_{task} = 0.1 and k_d = 1
		100 tasks, 20 CNs	V_{CN} = 0.1, V_{task} = 0.1 and k_d = (1, 1.3,1.8)
		1,000 tasks, 20 CNs	V_{CN} = 0.1, V_{task} = 0.1 and k_d = 1
		1000 tasks, 20 CNs	V_{CN} = 0.35, V_{task} = 0.35 and k_d = 1.8
2.	Medium sized workloads	10,000 tasks, 20 CNs	V_{CN} = 0.1, V_{task} = 0.35 and k_d = 1
		10,000 tasks, 20 CNs	V_{CN} = 0.1, V_{task} = 0.1 and k_d = 1
3.	Large sized workloads	100,000 tasks, 20 CNs	V_{CN} = 0.1, V_{task} = 0.1 and k_d = 1

Introducing the DTVS scheme in all of the heuristics rendered lower mean energy consumption. The DTVS scheme improves mean energy consumption for all of the heuristics by 31.09 %. The results using the DTVS for 100 tasks are depicted in Fig. 6c. The UtyFunc reported lower minimum and maximum energy consumption values and had highest standard deviation. These simulations were executed for a tighter deadline scaling parameter k_d, leaving less slack to be utilized for DTVS. The G-Min, G-Max, and G-Deadline have similar results with minimum mean energy consumption for scheduling 100 tasks. All heuristics using the DTVS had improved average energy consumption as compared to the results without using the DTVS.

The tasks having shortest execution times are allocated first when G-Min greedy heuristic is employed, while G-Max heuristic schedules those tasks first that have longest execution times. The tasks with lowest deadline d_i are scheduled first when G-Deadline is employed. The results for energy consumption for these heuristics are compared and depicted in Fig. 7a and b with different values of deadline scaling parameter k_d. The lower the value of the deadline scaling parameter k_d, tighter are the deadlines for the tasks.

The mean energy consumption with and without using the DTVS decreases when deadline scaling parameter (k_i) is varied from 1 to 1.3, and mean energy consumption slightly increases when k_d is varied from 1.3 to 1.8. From aforementioned discussion, it can be inferred that when k_d is increased from 1 to 1.3, then mean energy consumption for G-Deadline, G-Min, and G-Max is decreased because of the fact that they make better use of the DVS and DTVS as compared to their counterparts. The MinMax heuristic reported large energy consumption because the tasks are scheduled to least efficient CNs first. On contrary, the mean energy consumption for these heuristics is slightly increased by varying k_d ($1.3 \leq k_d \leq 1.8$) because of the fact that the use of DVS for very loose deadlines results in very large makespan.

The energy consumed by each heuristic is calculated by taking product of makespan and the instantaneous power of the CN to which the task is scheduled. Therefore, the mean energy consumption with loose deadlines is increased.

1,000 Tasks: Figure 8a, b, and c compare makespan, energy consumption without using DTVS, and energy consumption with using DTVS, for 1000 tasks, respectively. The GenAlgo outperformed all of the other heuristics in terms of makespan because it did not employ DVS and utilized the voltage at maximum available level. The MinMax schedules the tasks on least efficient CNs first. The MinMax and UtyFunc have large makespan while scheduling 1000 tasks. The ObjFunc and GenAlgo-DVS both have a mean makespan almost equal to the grand mean of all of the heuristics. The results for energy consumption for the G-Max, G-Deadline, and G-Min, are almost similar as they were for 100 tasks with G-Min having lowest value of mean energy consumption (1.6%) than the G-deadline and G-Max. The GenAlgo produced better results for scheduling 1000 tasks as compared to scheduling 100 tasks due to its convergence for large sized workloads. When the DTVS is employed, MinMax revealed better performance as compared to the mean energy consumption without using the DTVS. The mean energy consumption with using the DTVS for all of the heuristics was improved by 17.65 % when compared to mean energy consumption with-

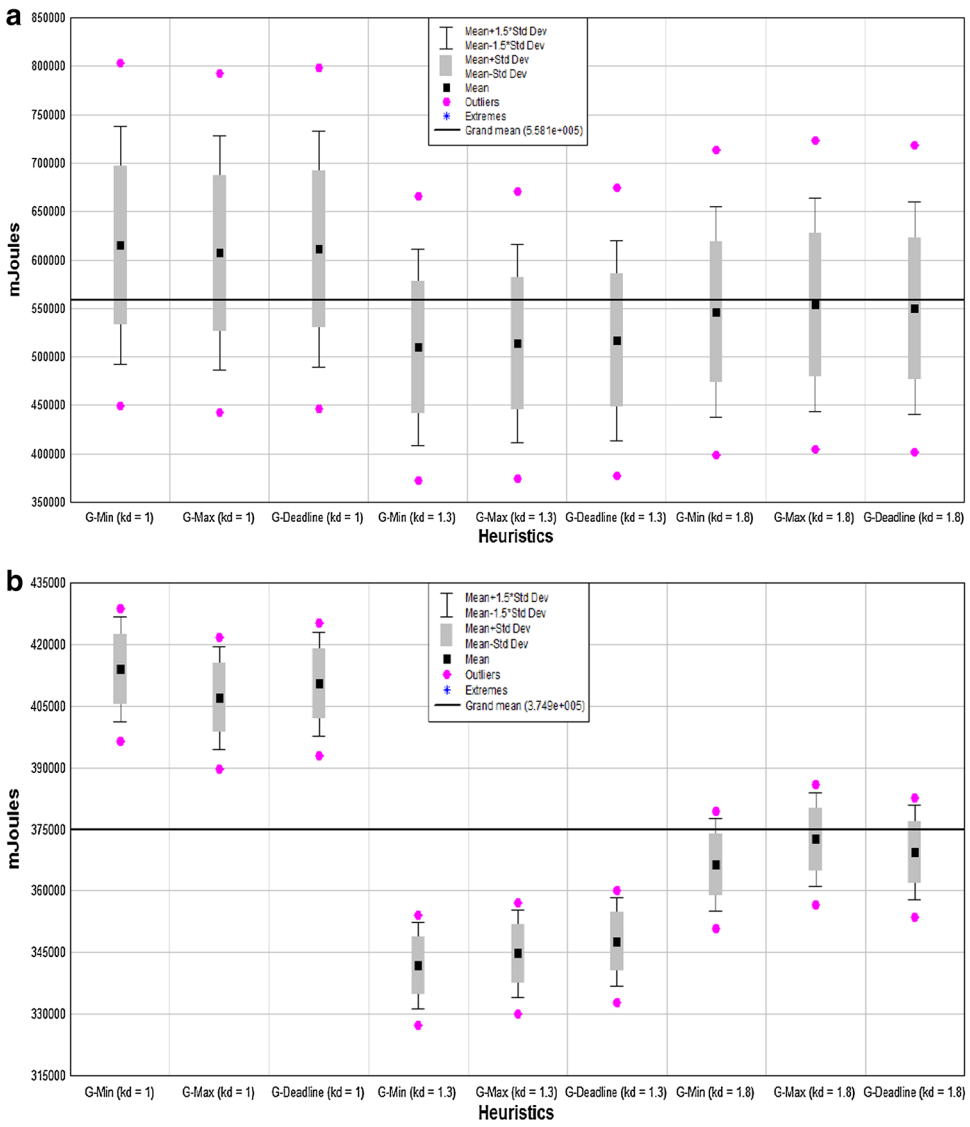


Fig. 7 Energy Consumption (without DTVS) for 100 tasks with various values of k_d Energy Consumption (with DTVS) for 100 tasks with various values of k_d

out using the DTVS. The performance of the GenAlgo was degraded when the DTVS was employed on all the heuristics. The performance of GenAlgo was degraded most when DTVS was employed amongst all the heuristics. This is due to the fact that the mean makespan for GenAlgo is lowest with low standard deviation resulting in less room for improvement.

The results for the G-Deadline, G-Min, and G-Max are compared without DTVS and with DTVS with high PE and task heterogeneity and loose deadline. The G-Deadline proved to the best amongst the three heuristics considered for the mean energy consumption without using the DTVS while scheduling 1,000 tasks. This is due to the fact that the G-Deadline heuristic schedules the tasks with shortest deadlines first and making the deadlines larger has little effect on its performance.

It can be observed from Fig. 9a that all of the three heuristics have lower minimum and larger maximum values for energy consumption that result in high standard deviation with high CN and task heterogeneity. There is an improvement of 46.60 % in terms of energy consumption when DTVS is employed in these heuristics with same system parameters as depicted in Fig. 9b. G-Max outperformed the other two heuristics and G-Deadline revealed worst performance. The heuristics performed well when DTVS is employed because of the diversity in CN speeds and task execution times as a result of increasing the task and CN heterogeneity.

5.2.2 Medium sized workloads

10,000 tasks: For medium sized workloads, we first compare the energy consumption of G-Deadline, G-Min, G-Max, and

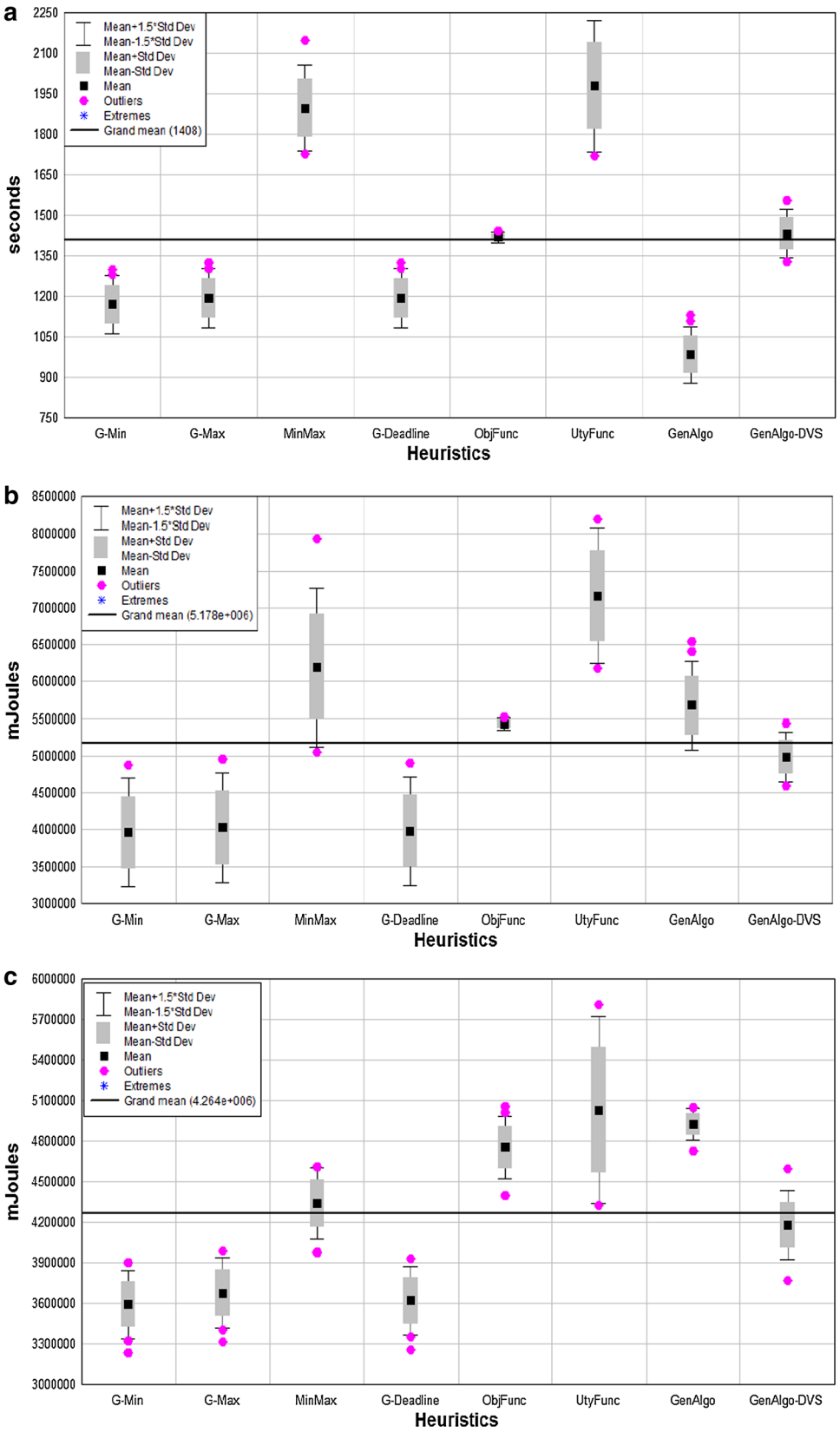


Fig. 8 Makespan for 1000 tasks (V_{task} = 0.1, V_{CN}=0.1, k_d= 1) Energy Consumption (without DTVS) for 1000 tasks (V_{task} = 0.1, V_{CN} = 0.1, k_d= 1) Energy Consumption (with DTVS) for 1000 tasks (V_{task} = 0.1, V_{CN} = 0.1, k_d = 1)

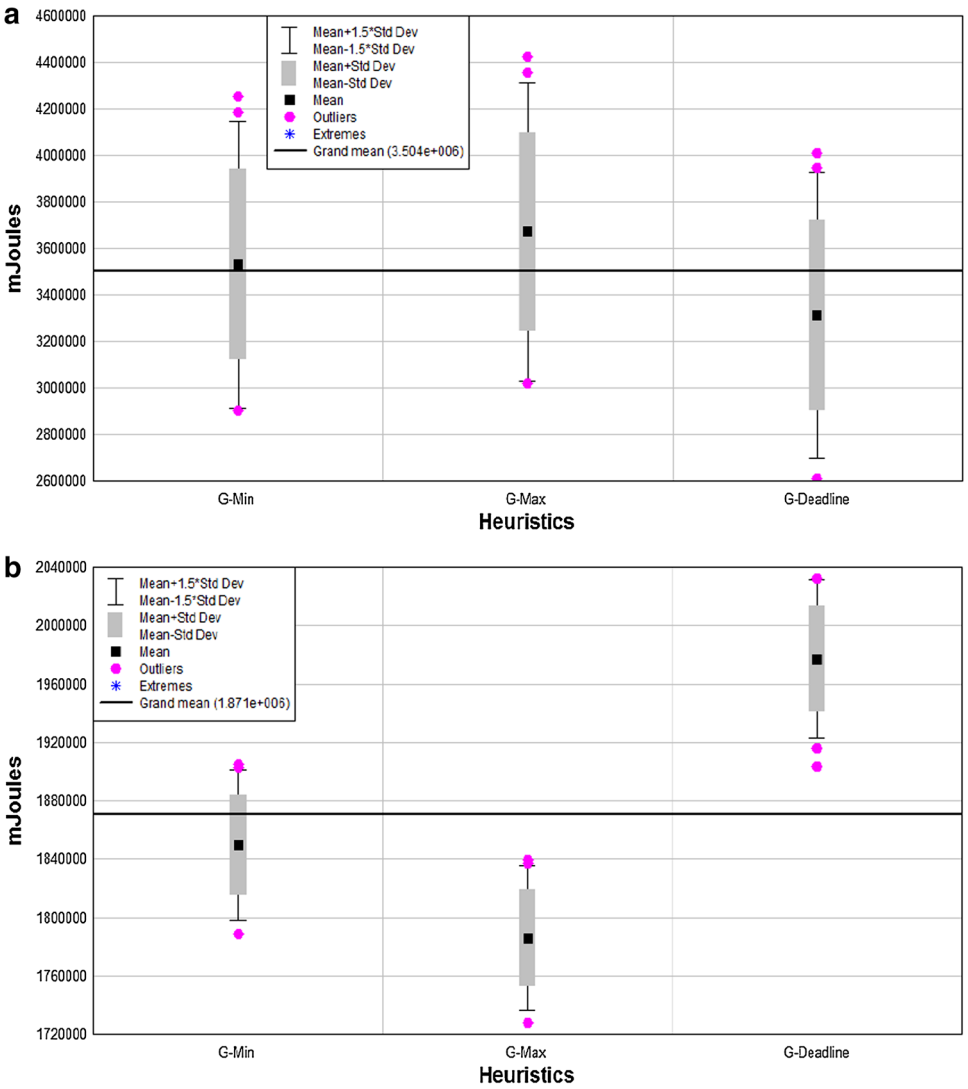


Fig. 9 a) Energy Consumption (without DTVS) for 1000 tasks (V_{task}=V_{CN}= 0.35, k_d= 1.8). b) Energy Consumption (with DTVS) for 1,000 tasks (V_{task} = V_{CN}= 0.35, k_d= 1.8)

ObjFunc for high task heterogeneity with low CN heterogeneity and tight deadline. The comparisons of mean energy consumption of these heuristics without employing DTVS and using DTVS are plotted in Fig. 10a, b, respectively.

Figure 10a reveals that ObjFunc produces better results with lower standard deviation and mean energy consumption. The G-Deadline has mean energy consumption almost equal to the grand mean of the above listed heuristics. G-Min performed worst among these heuristics consuming 10.10 % more energy than ObjFunc for scheduling 10,000 tasks.

The G-Min revealed significant improvement in energy consumption as compared to the other three heuristics, when DTVS was employed. There is a large slack in schedules of G-Min due to the fact that G-Min schedules the shortest tasks first.

The DTVS utilizes these slacks for improvement in the mean energy consumption. The G-Max schedules largest tasks first, leaving small tasks at the end to be scheduled. Therefore, small slack in schedules impose a limitation on performance of DTVS for G-Max. An improvement of 27.18 % is observed in the mean energy consumption by using DTVS in these four heuristics.

When scheduling 10,000 tasks we compared the results of the G-Max, G-Min, MinMax, G-Deadline, ObjFunc, and UtyFunc for makespan and energy consumption with and without employing DTVS in Fig. 11a–c, respectively. The G-Max, G-Deadline, and G-Min have the lowest makespan and are similar to each other. The MinMax has slightly higher makespan than the G-Max, G-Deadline, and G-Min. ObjFunc and UtyFunc produced worst results in terms of makespan (please see Fig. 11a).

The mean energy consumption for scheduling 10,000 tasks without deploying DTVS is depicted in Fig. 11b. Figure 11c shows the mean energy consumption with DTVS

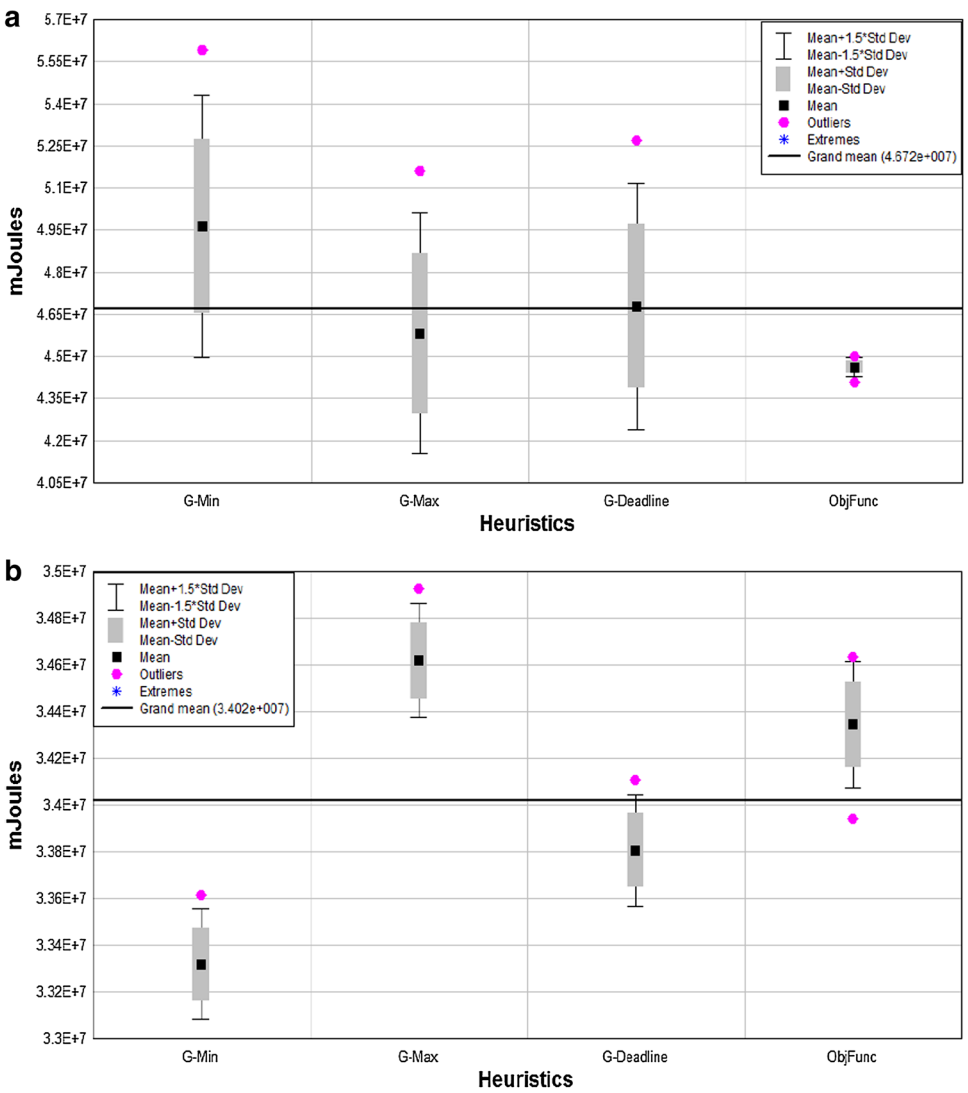


Fig. 10 **a** Energy Consumption (with DTVS) for 10,000 tasks (V_{CN} = 0.1, k_d = 1, V_{task} = 0.35), **b** Energy Consumption (with DTVS) for 10,000 tasks (V_{CN} = 0.1, k_d = 1, V_{task} = 0.35)

scheme. By comparing Fig. 11b and c, it is revealed that a significant improvement in the mean energy consumption of MinMax is achieved when DTVS is used. There is a decrease of 10.98 % in the mean energy consumption of all of the heuristics with the deployment of DTVS. There is an improvement of 31.71 % in mean energy consumption of the MinMax heuristic, which is significant when considering 10.98 % improvement for all of the heuristics collectively.

5.2.3 Large sized workloads

100,000 tasks: The ObjFunc produced best results in terms of mean energy consumption and mean makespan for large sized workloads, because ObjFunc considers multiple tasks and multiple CNs in its objective function when assigning the tasks to the CNs. The results for makespan and energy consumption (without DTVS) for large sized workloads are depicted in Fig. 12a and b, respectively. UtyFunc produced results with large makespan and mean energy consumption because it considers the relative importance of speed and the execution time of the tasks, simultaneously. This results in a tradeoff between makespan and energy consumption, failing to produce better results. The G-Max, G-Deadline, and G-Min, all produced similar results with lower value of mean energy consumption and mean makespan than the grand mean.

UtyFunc revealed significant improvement in the mean energy consumption when DTVS was used. Overall 31.53 % reduction in mean energy consumption of all of the heuristics is observed by using DTVS scheme. The results for mean energy consumption with DTVS are depicted in Fig. 12c.

On the basis of the results obtained in this section we can safely draw a conclusion that irrespective of the work-

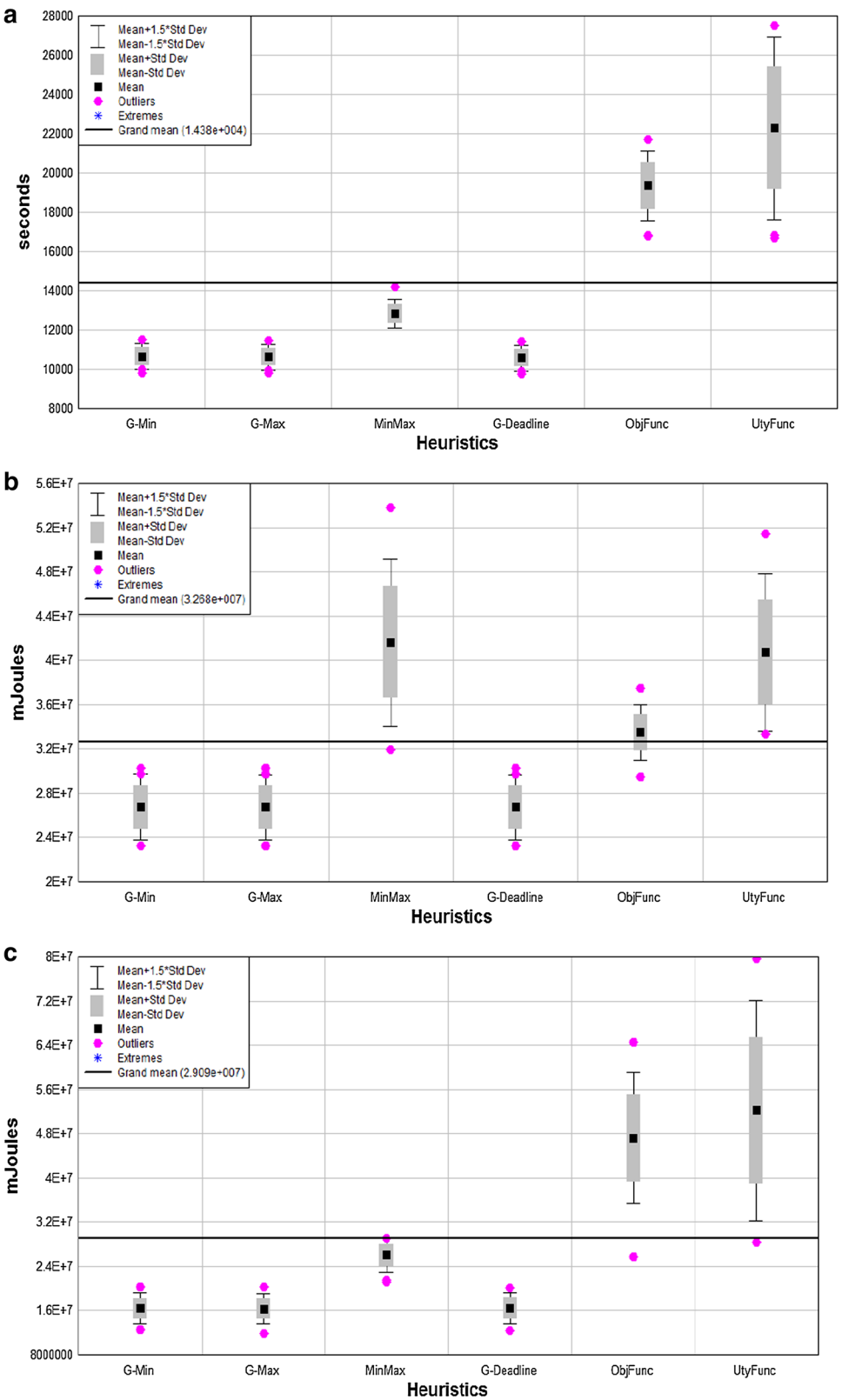


Fig. 11 Makespan for 10,000 tasks (V_{task}= V_{CN}=0.1, k_d= 1) Energy Consumption (without DTVS) for 10,000 tasks (V_{task}= V_{CN}= 0.1, k_d= 1) Energy Consumption (with DTVS) for 10,000 tasks (V_{task}= V_{CN}= 0.1, k_d= 1)

load size the heuristics G-Deadline, G-Min, and G-Max exhibits better results in terms of energy savings. If we categorize the workload size we observe that the GenAlgo-DVS reported better results for small sized workload. Nevertheless, for large sized workload ObjFunc produces promising results.

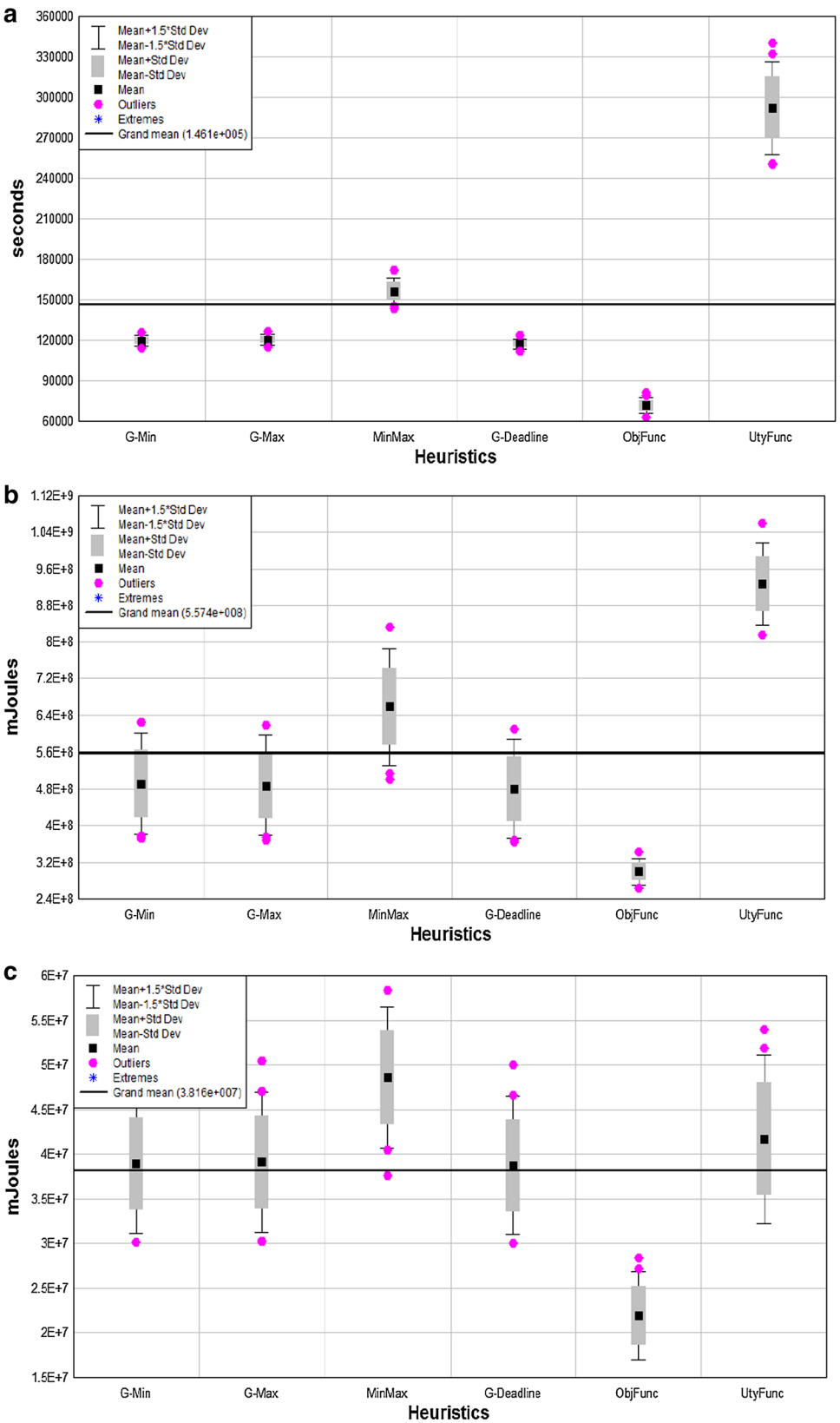


Fig. 12 Makespan for 100,000 tasks (V_{task}= **V_{CN}= 0.1, k_d**= 1) Energy Consumption (without DTVS) for 100,000 tasks(V_{task}= V_{CN}= 0.1, k_d= 1) Energy Consumption (with DTVS) for 100,000 tasks (V_{task}= V_{CN}= 0.1, k_d= 1)

6 Conclusions

With the growing demand of HPC for heavy computations, the power cost of the aforementioned is acting as a limiting factor and needs to be controlled and overcome using power-aware system solutions. This paper underlines the role of voltage level of nodes in cluster's power consumption and presents a methodology, termed as PAJS, that combines energy-efficient job scheduling with node awareness. We have analyzed and compared eight task scheduling techniques. The role of power optimization in modern HPC is emphasized and the proposed PAJS approach optimizes the tradeoff between energy efficient nodes (to reduce the amount of energy consumed) and performance aware patterns (to minimize the makespan). G-Max, G-Deadline and G-Min performed better for all sized workloads as compared to other heuristics. The minimum overall reduction in the energy consumption using DTVS was 10.98 %. Nevertheless, the maximum energy savings using DTVS is 31.71 % as compared to when DTVS was not employed. For small-sized workloads GenAlgo-DVS also yielded better results in terms of mean makespan and mean energy consumption. However, for large-sized workloads ObjFunc performed well in addition to G-Max, G-Deadline, and G-Min.

The simulation results approve the superior performance of all the heuristics of the proposed methodology (PAJS) in terms of reduction in the makespan and energy consumption of the nodes comprising the HPC. The work presented here is not just restricted to clusters but can easily be adapted to grids, workstations, and datacenters. Both on the practical and theoretical point of view, we have introduced a coherent framework for the optimization of power in various resource scheduling strategies in HPC.

Acknowledgments The authors would like to thank Saif-ur-Rehman and Mazhar Ali for their valuable reviews, suggestions, and comments.

References

1. Abbas, A., Ali, M., Fayyaz, A., Ghosh, A., Kalra, A., Khan, S.U., Khan, M.U.S., Menezes, T.D., Pattanayak, S., Sanyal, A., Usman, S.: A survey on energy-efficient methodologies and architectures of network-on-chip. Comput. Electr. Eng. doi:10.1016/j.compeleceng.2014.07.012
2. Ahmad, I., Ranka, S., Khan, S.U.: Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In: 22nd IEEE International parallel and distributed processing symposium, pp. 1–6 (2008)
3. Alfonso, C.D., Caballer, M., Avarruiz, F., Hernandez, V.: An energy management system for cluster infrastructures. J. Comput. Electr. Eng. **39**(8), 2579–2590 (2013)
4. Ali, S., Siegel, H.J., Maheswaran, M., Hensgen, D., Ali, S.: Task execution time modeling for heterogeneous computing systems. In: IEEE 9th heterogeneous computing workshop, pp. 185–199 (2000)
5. Ali, S., Siegel, H.J., Maheswaran, M., Hensgen, D., Ali, S.: Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang J. Sci. Eng. **3**(3), 195–207 (2000)
6. Aziz, M.A., Khan, S.U., Loukopoulos, T., Bouvry, P., Li, H., Li, J.: An overview of achieving energy efficiency in on-chip networks. Int. J. Commun. Netw Distrib. Syst. **5**(4), 444–458 (2010)
7. Beloglazov, A., Abawaj, J., Buyya, R.: Energy aware resource allocation heuristics for efficient management of data centers for cloud computing. Futur Gener. Comput. Syst. **28**(5), 755–768 (2012)
8. Chaparro-Baqueero, G.A., Zhou, Q., Liu, C., Tang, J., Liu, S.: Power-efficient schemes via workload characterization on the Intel's single chip cloud computer. In: IEEE parallel and distributed processing symposium workshops and Ph.D. forum (IPDPSW), pp. 999–1006 (2012)
9. Al-Daud, H., Al-Azzonib, I., Down, D.G.: Power aware linear programming based scheduling for heterogeneous computer clusters. In: Future generation computer system, vol. **24**, 5th edn, pp. 745–754, May 2012. [Special section: energy efficiency in large scale distributed system]
10. Diaz, C.O., Guzek, M., Pecero, J.E., Bouvry, P., Khan, S.U.: Scalable and energy-efficient scheduling techniques for large-scale systems. In: International conference on computer and information technology (CIT '11), pp. 641–647 (2011)
11. Huang, S., Feng, W.: Energy efficient cluster computing via accurate workload characterization. In proceeding of the 2009 9^{th} IEEE/ACM international symposium on cluster computing and the grid. CCGRID, IEEE S (2009)
12. Andersson, J.: A survey of multiobjective optimization in engineering design. Technical report, Department of Mechanical Engineering, Linköping University, Linköping, Sweden (2000)
13. Khan S.U., Ardil, C.: A game theoretical energy efficient resource allocation technique for Large distributed computing systems. In: International conference on parallel and distributed processing, techniques and applications (PDPTA), pp. 48–54 (2009)
14. Khan, S.U., Ardil, C.: On the joint optimization of performance and power consumption in data centers. In: International conference on distributed, high-performance and grid computing, pp. 660–666 (2009)
15. Khan, S.U., Min-Allah, N.: A goal programming based energy efficient resource allocation in data centers. J. Supercomput. **61**(3), 502–519 (2012)
16. Kliazovich, D., Arzo, S.T., Granelli, F., Bouvry, P., Khan, S.U.: Accounting for load variation in energy efficient data centers. In: IEEE international conference on communications (ICC), pp. 1154–1159 (2013)
17. Kolodziej, J., Khan, S.U., Wang, L., Byrski, A., Min-Allah, N., Madani, S.A.: Hierarchal genetic based grid scheduling with energy optimization. J. Clust. Comput. **16**(3), 591–609 (2013)
18. Kolodziej, J., Khan, S.U., Wang, L., Kisiel-Dorohinicki, M., Madani, S.A., Niewiadomska-Szynkiewicz, E., Zomaya, A.Y., Xu, C.-Z.: Security, energy, and performance-aware resource allocation mechanisms for computational grids. Futur Gener. Comput. Syst. **31**, 77–92 (2014)
19. Krioukov, A., Goebel, C., Alspaugh, S., Chen, Y., Culler, D.E., Katz, R.H.: Integrating renewable energy using data analytics systems: challenges and opportunities. Bull. IEEE Comput. Soc. Tech. Comm. **34**(1), 3–11 (2011)
20. Lang, W., Harizopoulos, S., Patel, J.M., Shah, M.A., Tsirogiannis, D.: Towards energy-efficient database cluster design. Proc. VLDB Endow. (PVLDB) **5**(11), 1684–1695 (2012)
21. Lang, W., Patel, J.M.: Energy management for map reduce cluster. Proc. VLDB **3**(1–2), 129–139 (2010)
22. Lindberg, P., Leingang, J., Lysaker, D., Khan, S.U., Li, J.: Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems. J. Supercomput. **59**(1), 323–360 (2010)

23. Lindberg, P., Leingang, J., Lysaker, D., Bilal, K., Khan, S.U., Bouvry, P., Ghani, N., Min-Allah, N., Li, J.: Comparison and analysis of greedy energy-efficient scheduling algorithms for computational grids. In: Zomaya, A.Y., Lee, Y.-C. (eds.) Energy aware distributed computing systems. Wiley, Hoboken (2012). ISBN 978-0-470-90875-4, Chapter 7
24. Maiuri, O.V., Moore, W.R.: Implications of voltage and dimension scaling on CMOS testing: the multidimensional testing paradigm. In: 16th IEEE symposium on VLSI test (1998)
25. Mehta, N., Amrutur, B.: Dynamic supply and threshold voltage scaling for CMOS digital circuits using in-situ power monitor. IEEE Trans. VLSI Syst. **20**(5), 892–901 (2012)
26. Min-Allah, N., Hussain, H., Khan, S.U., Zomaya, A.Y.: Power efficient rate monotonic scheduling for multi-core systems. J. Parallel Distrib. Comput. **72**(1), 48–57 (2012)
27. Shuja, J., Madani, S.A., Bilal, K., Hayat, K., Khan, S.U., Sarwar, S.: Energy efficient data centers. Computing **94**(12), 973–994 (2012)
28. Sha, S., Zhou, J., Liu, C., Quan, G.: Power and energy analysis on Intel single-chip cloud computer system. In: Proceedings of IEEE South Easton, pp. 1–6 (2012)
29. Usman, S., Khan, S.U., Khan, S.: A comparative study of voltage/frequency scaling in NOC. 2013. In: IEEE International conference on electro/information technology (2013)
30. Valentini, G.L., Lassonde, W., Khan, S.U., Min-Allah, N., Madani, S.A., Li, J., Zhang, L., Wang, L., Ghani, N., Kolodziej, 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. **16**(1), 3–15 (2011)
31. Wang, L., Khan, S.U., Chen, D., Kołodziej, J., Ranjan, R., C.Z., Xu, Zomaya, A.: Energy aware parallel task scheduling in a cluster. Futur. Gener. Comput. Syst. **29**(7), 1661–1670 (2013)
32. Zong, Z., Qin, X., Ruan, X., Bellam, K., Nijim, M., Alghamdi, M.: Energy-efficient scheduling for parallel applications running on heterogeneous clusters. In: International conference on parallel processing (ICPP 2007), pp. 19–26 (2007)

Kashif Bilal is an Assistant Professor at COMSATS Institute of Information Technology, Abbottabad, Pakistan. He received his PhD in Electrical and Computer Engineering from North Dakota State University, Fargo, USA in 2014. He also received College of Engineering (CoE) Graduate Student Researcher of the year 2014 award at NDSU. His research interests include energy efficient high speed networks, Green Computing, and robustness in data centers. Currently, he is focusing on exploration of network traffic patterns in real data centers and development of the data center network workload generator.

Ahmad Fayyaz received a BS degree from COMSATS Institute of Information & Technology, Abbottabad, Pakistan, and MS from University of Surrey, Guildford, UK. Currently, he is a PhD Student in North Dakota State University, Fargo, ND, USA. Ahmad Fayyaz research interests include energy efficient scheduling in clusters, cloud computing and distributed systems, and system optimization for energy efficiency.

Samee U. Khan is an associate professor of electrical and computer engineering at the North Dakota State University. Samee's research interests include optimization, robustness, and security of: cloud, grid, cluster and big data computing, social networks, wired and wireless networks, power systems, smart grids, and optical networks. He maintains the GreenCloud simulator and the CloudNetSim++ simulator. He is an associate editor of the IEEE Transactions on Computers, IEEE Access, IEEE Cloud Computing, IEEE Communications Surveys and Tutorials, IEEE IT Pro, Scalable Computing, Cluster Computing, Security and Communication Networks, and International Journal of Communication Systems.

Saeeda Usman received a Master of Science in Advanced Electrical & Electronic Engineering degree from University of Leicester, UK. Currently, she is pursuing PhD at North Dakota State University, Fargo, USA. Her research interests encompasses topics related to Energy efficiency, performance optimization, and robustness improvement in Cloud Computing.