

Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems

Peder Lindberg · James Leingang ·
Daniel Lysaker · Samee Ullah Khan · Juan Li

© Springer Science+Business Media, LLC 2010

Abstract In this paper, we study the problem of scheduling tasks on a distributed system, with the aim to simultaneously minimize energy consumption and makespan subject to the deadline constraints and the tasks' memory requirements. A total of eight heuristics are introduced to solve the task scheduling problem. The set of heuristics include six greedy algorithms and two naturally inspired genetic algorithms. The heuristics are extensively simulated and compared using an simulation test-bed that utilizes a wide range of task heterogeneity and a variety of problem sizes. When evaluating the heuristics, we analyze the energy consumption, makespan, and execution time of each heuristic. The main benefit of this study is to allow readers to select an appropriate heuristic for a given scenario.

Keywords Distributed systems · Energy-efficiency · Optimization · Large-scale systems

P. Lindberg · J. Leingang · D. Lysaker · S.U. Khan (✉) · J. Li
NDSU-CIIT Green Computing and Communications Laboratory, North Dakota State University,
Fargo, ND 58108-6050, USA
e-mail: samee.khan@ndsu.edu

P. Lindberg
e-mail: peder.lindberg@ndsu.edu

J. Leingang
e-mail: james.leingang@ndsu.edu

D. Lysaker
e-mail: daniel.lysaker@ndsu.edu

J. Li
e-mail: j.li@ndsu.edu

1 Introduction

A large-scale distributed system is often used as a centralized repository for storage and management of (user) data and information [1]. Energy consumption is widely recognized to be a critical problem in large-scale distributed systems, such as a computational grid. As the demand for internet services increases, so does the amount of electricity used to power large-scale distributed systems [2].

Dynamic power management (DPM) [3] and dynamic voltage scaling (DVS) [4] are two techniques that can be used to reduce energy consumption in large-scale distributed systems. The DPM technique is a design methodology used to decrease the energy consumption of a processing element (PE) in each machine by dynamically powering down PEs. Normally when a PE does not have a full workload, the PE will become idle. With DPM, the PE will be turned off instead. Turning a PE on or off has a high transition cost. With the DVS approach, each PE's supply voltage (V_{dd}) can be scaled to a discrete number of V_{dd} levels. By decreasing V_{dd} and operational frequency (f) of a PE, the amount of energy consumed may be reduced. A PE completes fewer computational cycles when running at a lower frequency; therefore, lowering the frequency increases the *makespan*. The *makespan* is defined as the time taken to execute all tasks received by the large-scale distributed system. The following equations give the relationship between f , power consumption, and energy consumption over the period $[0, T]$:

$$f = \frac{k \cdot (V_{dd} - V_t)^2}{V_{dd}}, \quad (1)$$

$$P = C_L \cdot N_{0 \rightarrow 1} \cdot f \cdot V_{dd}^2, \quad (2)$$

$$E = \int_0^T P(t) dt, \quad (3)$$

where C_L is the switching capacitance, $N_{0 \rightarrow 1}$ is the switching activity, k is a constant that is dependent on the circuit, T is the total time, and V_t is the circuit threshold voltage.

The energy consumption can be reduced by lowering f as given in (1). However, when f is lowered, the execution time of the task increases, which may cause the task to violate the deadline constraint. When scheduling tasks to a single PE, an effective technique for reducing the energy consumption would be lowering f until the task can no longer meet the deadline constraint. Because the PE-task pair that results in the minimum energy consumption is not known, the use of multiple PEs cause the problem to become much more complicated.

Most DPM techniques utilize instantaneous power management features supported by hardware. For example, in [5], the operating system's power manager is extended by an adaptive power manager. This adaptive power manager uses the processor's DVS capabilities to reduce or increase the CPU frequency, thereby minimizing the total energy consumption [6]. The DVS technique combined with a turn on/off technique is used to achieve high-power savings while maintaining all deadlines in [7]. In [8], a scheme to concentrate the workload on a limited number of

processors is introduced. This technique allows the rest of the processors to remain switched off for a longer time.

There are a wide variety of power management techniques, such as heuristic-based approaches [1, 2, 9–11], genetic algorithms [12–15], and constructive algorithms [16]. Most of these techniques have been studied using relatively small sets of tasks. The techniques introduced in this paper were given large sets of tasks allowing one to compare and analyze some traditional power management techniques when applied to large-scale distributed systems.

For a thorough overview of previously published results, readers are encouraged to read surveys, such as [17, 18], and [19].

In this paper, we study the energy-aware task allocation (EATA) problem for assigning a set of tasks to a set of PEs equipped with DVS modules. To circumvent the above mentioned problems, we propose the following four-step energy minimization methodology (4EMM):

1. **Resource Allocation:** Determine the number and type of PEs that form a suite to execute a given number of tasks.
2. **Resource Matching:** Select a PE-task pair that can fulfil user specified run-time constraints.
3. **Resource Scheduling:** Determine the order and the corresponding DVS level for each PE-task pair.
4. **Solution Evaluation:** Calculate the dollar cost of the resource allocation (Step 1) and energy consumption of the resource scheduling (Step 3).

This paper compares and analyzes eight heuristics-based task scheduling algorithms. This set of heuristics uses a wide variety of techniques, such as iterative, greedy, constructive, and genetic algorithms. Such heuristics can be a plausible solution to the EATA problem studied in this paper. In all of the proposed heuristics, the assumptions and system parameters (number of tasks, task deadlines, PE dollar cost, etc.) are kept the same to maintain a fair comparison. The studied heuristics were modeled after two major classes of algorithms. Six heuristics are greedy based heuristics, namely G-Min, G-Max, G-Deadline, MaxMin, ObFun, and UtFun. The final two heuristics, GenAlg and GenAlg-DVS, are naturally inspired genetic algorithms.

We will compare and analyze the above techniques by examining the results of numerous simulations. To incorporate variance in our simulations, we vary the task and PE heterogeneity. We also vary the number of tasks between 100 and 100,000. Varying the number of tasks in our workload allows us to determine which heuristics produce better results for different sized problems. A detailed explanation of our simulation test-bed will be given in the subsequent text.

The remainder of this paper is organized as follows. The problem formulation is introduced in Sect. 2. We discuss the resource allocation and task scheduling heuristics in Sect. 3. The simulation results are reviewed in Sect. 4. Finally, we present a conclusion in Sect. 5.

2 Problem formulation

The most frequently used acronyms in this paper are listed in Table 1.

Table 1 Notations and their meanings

| Symbols | Meaning | Symbols | Meaning |
|------------------|--|-----------------------|--|
| DVS | Dynamic voltage scaling | G-Min | Greedy Heuristic that schedules shortest tasks first |
| DPM | Dynamic power management | G-Max | Greedy Heuristic that schedules longest tasks first |
| EATA | Energy-aware task allocation | G-Deadline | Greedy Heuristic that schedules tasks with the most urgent deadline first |
| CVB | Coefficient of variation based methodology | MaxMin | Greedy Heuristic that initially schedules tasks to the least efficient PEs |
| 4EMM | 4-step energy minimization methodology | ObFun | Greedy Heuristic that uses two objective functions to determine task assignments |
| PE | Processing element | UtFun | Greedy Heuristic schedules tasks based on a utility function |
| V_{dd} | PE supply voltage | GenAlg | Genetic Algorithm |
| M | Makespan | GenAlg-DVS | Genetic Algorithm that utilizes DVS |
| T | Set of all t_i | c_i | Computational cycles required by t_i |
| t_i | i th task $\in T$ | $I-ETC$ | Indexes for ETC Matrix |
| d_i | Deadline of t_i | t_{ij} | Run-time of t_i on PE_j |
| m_{t_i} | Memory requirement of t_i | t_{ijk} | Run-time of t_i on PE_j at DVS_k |
| m_{PE_j} | Memory available to PE_j | m_j | Run-time of PE_j |
| DVS_k | k th DVS level | k_{idle} | Power scalar for idle PE |
| EEC | Estimated energy consumption | E_{idle} | Energy consumed while PE is idle |
| ETC | Estimated time of completion | μ_{task} | Average task execution time |
| \mathcal{PE} | Set of PEs in PE allocation | V_{task} | Variance in task execution time |
| \mathcal{PE}_p | Set of PEs in PE pool | V_{PE} | Variance in PE heterogeneity |
| PE_j | j th PE $\in \mathcal{PE}$ | GAP | Generalized Assignment Problem |
| D_j | Dollar cost of PE_j | $N_{0 \rightarrow 1}$ | Switching activity |
| \mathcal{D} | Total dollar cost constraint | | |
| p_j | Power consumption of PE_j | | |
| E_j | Energy consumed by PE_j | | |
| p_{ij} | Energy consumed by PE_j to execute t_i | | |

2.1 The system model

Consider a large-scale distributed system that is a set of tasks (referred to as a meta-task) and a collection of PEs.

PEs. Let the set of PEs be denoted as, $\mathcal{PE} = \{PE_1, PE_2, \dots, PE_m\}$. Each PE is assumed to be equipped with a DVS module. We assume that the transition time between any two DVS levels is constant and negligible for the problem considered in this paper. Such an assumption was also previously made in [7, 18, 20, 21], and [13]. A PE is characterized by:

- The instantaneous power consumption of the PE, p_j . Depending on the PE's DVS level, p_j may vary between p_j^{\min} to p_j^{\max} , where $0 < p_j^{\min} < p_j^{\max}$.
- The available memory of PE, m_{PE_j} .

- The dollar cost of the PE, D_j . The set of PEs must not cost more than the total system dollar cost constraint, D .

Tasks. A metatask, $T = \{t_1, t_2, \dots, t_n\}$, is a set of tasks where t_i is a task. Each task is characterized by:

- The number of computational cycles, c_i , that need to be completed.
- The memory requirement of a task, m_{t_i} .
- The deadline, d_i , which is the time that a task must finish.

Preliminaries. Suppose we are given a set of PEs and a metatask, T . Each $t_i \in T$ must be mapped to a PE such that the deadline constraint of t_i is fulfilled. That is, the run-time of PE_j must be less than d_i . Let the run-time of PE_j be denoted by m_j . A feasible task to PE mapping occurs when each task in the metatask can be mapped to at least one PE_j while satisfying all of the associated task constraints. If $m_{PE_j} < m_{t_i}$, then t_i cannot be executed on PE_j .

2.2 Formulating the energy-makespan minimization problem

Given is a set of PEs and a metatask, T . *The problem can be stated as:*

- *The total energy consumed by the PEs is minimized.*
- *The makespan, M , of the metatask, t , is minimized.*

We can say mathematically,

$$\text{minimize } \sum_{i=1}^n \sum_{j=1}^m p_{ij} x_{ij} \quad \text{and} \quad \text{minimize } \max \sum_{i=1}^n t_{ij} x_{ij}$$

subject to the following constraints:

$$x_{ij} \in 0, 1, \quad i = 1, 2, \dots, n; \quad j = 1, 2, \dots, m, \tag{4}$$

$$t_i \rightarrow m_j, \quad \forall i, \forall j; \quad \text{if } m_{PE_j} > m_{t_i}; \quad \text{then } x_{ij} = 1, \tag{5}$$

$$t_{ij} x_{ij} \leq d_i, \quad \forall i, \forall j, x_{ij} = 1, \tag{6}$$

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

$$\prod_{i=1}^n (t_{ij} x_{ij} \leq d_i) = 1, \quad \forall i, \forall j, x_{ij} = 1, \tag{8}$$

$$\sum_{j=1}^m D_j \leq D, \quad x_{ij} = 1. \tag{9}$$

Constraint 4 is the mapping constraint. t_i is assigned to PE_j when $x_{ij} = 1$. Constraint 5 elaborates on this mapping in conjunction to the memory requirements and states that a mapping can exist only if PE_j has enough memory to execute t_i . Constraint 6 relates to the fulfilment of the deadline of each task. Constraint 7 shows

there is a Boolean relationship between the deadline and the actual execution time of the tasks. Constraint 8 relates to the deadline constraints of the metatask that will hold if and only if the deadline, d_i , for each $t_i \in T$ is satisfied. Constraint 9 pertains to the total dollar cost constraint, \mathcal{D} .

The EATA problem formulation is a multi-constrained, multiobjective optimization problem. The preference must be given to one objective over the other because the optimization of energy and M oppose each other. The formulation is in the same form as the Generalized Assignment Problem (GAP) except for Constraints (6, 7, 8, and 9). The major difference between GAP and EATA is that the capacity of resources in GAP, in terms of the utilization of instantaneous power, is defined individually, whereas in EATA the capacity of resources is defined in groups [22].

3 Proposed algorithms

In this section, we will describe the inner workings of our eight proposed heuristics. Figure 1 illustrates the relationship between 4EMM and the proposed heuristics.

All of the task execution times are obtained from an estimated time of completion (ETC) matrix [23]. An ETC matrix is a 2-d array with $|T|$ rows and $|\mathcal{PE}_p|$ columns. Each element in the ETC matrix corresponds to an execution time of t_i on PE_j , where i is the row and j is the column. To generate the ETC matrix, we use a coefficient-of-variation based (CVB) ETC matrix generation method [24]. There are three major parameters that determine the heterogeneity of the ETC matrix:

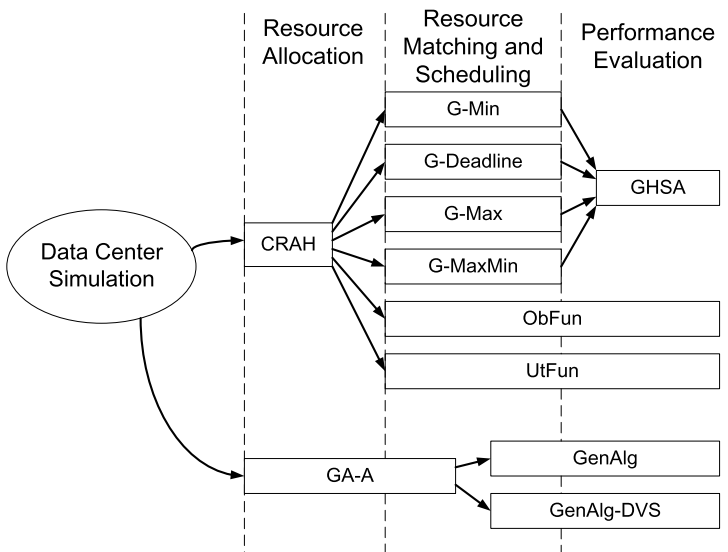


Fig. 1 Simulation flow chart

1. The average execution time of each $t_i \in T$, μ_{task} .
2. The variance in the task execution time, V_{task} .
3. The variance in the PE heterogeneity, V_{PE} .

Because CVB uses a gamma distribution [25], the characteristic shape parameter, α , and scale parameter, β , must be defined. The gamma distribution's parameters, α_{task} , α_{PE} , β_{task} , and β_{PE} can be interpreted in terms of μ_{task} , V_{task} , and V_{PE} . For a gamma distribution, $\mu = \beta\alpha$ and $V = 1/\sqrt{\alpha}$. Then

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

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

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

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

where $G(\alpha_{\text{task}}, \beta_{\text{task}})$ is a number sampled from a gamma distribution.

The d_i for each t_i is derived from the ETC matrix and can be represented by

$$d_i = \frac{|t_i|}{|\mathcal{PE}|} \cdot \arg_j \max(t_{ij}) \cdot k_d, \tag{14}$$

where k_d is a parameter that can tighten d_i [26, 27].

3.1 Greedy heuristics

The resource allocation (Step 1 of 4EMM) for the following six greedy heuristics is achieved by CRAH. Algorithm 1 shows the pseudo-code for CRAH. The CRAH algorithm takes as inputs an ETC matrix, \mathcal{PE}_p , and d_i for all $t_i \in T$. The output of CRAH is the T to \mathcal{PE} mapping, the energy consumption of the best solution, \mathcal{E}_{\min} , and M . At Line 1, one of the six greedy heuristics is invoked to *rearrange* the ETC matrix in the order the tasks will be scheduled. This step is different for each heuristic. Figure 2 illustrates one method of *rearranging* the ETC matrix. Figure 2(a) shows the original ETC matrix. The rows are sorted in ascending order (Fig. 2(b)). Next, the rows of the ETC matrix are swapped such that the execution times in the first column are arranged in ascending order (Fig. 2(c)). Because one must maintain indexing for a given ETC matrix, under each operation, we maintain the associated index with each element of the matrix. For the above mentioned matrix *rearranging* procedures, the corresponding index matrices (*I-ETC*'s) are shown in Figs. 2(d)–(f). Next, an estimated energy consumption (EEC) matrix is generated by multiplying the PE's instantaneous power consumption by the tasks' estimated completion time.

The outer **while** loop (Line 3) repeats until there is no significant improvement in solution quality. Let k be the number of loops with no improvement. The solution is considered sufficient when $k \geq k_{\max}$. An initial resource allocation, \mathcal{PE} , is generated

Input: $ETC, \mathcal{PE}_p, d_i \forall t_i \in T$
Output: T to \mathcal{PE} mapping, \mathcal{E}_{\min}, M

- 1 INVOKE Greedy Heuristic to *rearrange ETC* and generate *EEC*;
- 2 **while** $k < k_{\max}$ **do**
- 3 Generate Random \mathcal{PE} ;
- 4 CALCULATE E_{sol} ;
- 5 $E_{\min} \leftarrow E_{\text{sol}}$;
- 6 **repeat**
- 7 $E'_{\min} \leftarrow E_{\min}$;
- 8 **foreach** $PE_j \in \mathcal{PE}_p$ **do**
- 9 Add PE_j to \mathcal{PE} ;
- 10 CALCULATE E_{sol} ;
- 11 **if** $E_{\text{sol}} < E_{\min}$ **then** $E_{\min} \leftarrow E_{\text{sol}}$ Remove PE_j from \mathcal{PE} ;
- 12 **end**
- 13 **foreach** $PE_j \in \mathcal{PE}$ **do**
- 14 Remove PE_j from \mathcal{PE} ;
- 15 CALCULATE E_{sol} ;
- 16 **if** $E_{\text{sol}} < E_{\min}$ **then** $E_{\min} \leftarrow E_{\text{sol}}$ Add PE_j to \mathcal{PE} ;
- 17 **end**
- 18 **if** $E_{\min} \geq \mathcal{E}_{\min}$ **then**
- 19 INCREMENT k
- 20 **else**
- 21 $k \leftarrow 0$;
- 22 $\mathcal{E}_{\min} \leftarrow E_{\min}$
- 23 **end**
- 24 **until** $E_{\min} \geq E'_{\min}$;
- 25 **end**

Algorithm 1: Constructive resource allocation heuristic (CRAH)

Fig. 2 ETC matrix *rearranged* for G-Min

| | | | | |
|---|--|---|--|---|
| $\begin{vmatrix} 8 & 7 & 10 \\ 10 & 9 & 5 \\ 6 & 12 & 7 \end{vmatrix}$ <p>(a)</p> | | $\begin{vmatrix} 7 & 8 & 10 \\ 5 & 9 & 10 \\ 6 & 7 & 12 \end{vmatrix}$ <p>(b)</p> | | $\begin{vmatrix} 5 & 9 & 10 \\ 6 & 7 & 12 \\ 7 & 8 & 10 \end{vmatrix}$ <p>(c)</p> |
|---|--|---|--|---|

ETC Matrices

| | | | | |
|--|--|--|--|--|
| $\begin{vmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{vmatrix}$ <p>(d)</p> | | $\begin{vmatrix} 2 & 1 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{vmatrix}$ <p>(e)</p> | | $\begin{vmatrix} 3 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{vmatrix}$ <p>(f)</p> |
|--|--|--|--|--|

Index Matrices

by randomly adding PEs until D is violated. Next, one of the six greedy heuristics is invoked to schedule the tasks, calculate M , and determine the energy consumption of this solution, E_{sol} .

Inside the **repeat-until** loop (Line 6), \mathcal{PE} is modified (Every $PE_j \in \mathcal{PE}_p$ is added and removed from \mathcal{PE}) until a locally optimal solution has been found. In Line 7, note that E'_{min} is the energy minima found in the previous iteration. In Lines 10 and 15, the solution is evaluated. The change to \mathcal{PE} that results in the largest decrease in E_{sol} is recorded as \mathcal{PE} . When a local energy minima is reached, E_{min} (i.e., CRAH can no longer add or remove a PE to decrease the energy consumption), the **repeat-until** loop terminates. If E_{min} is less than the global energy minimum, \mathcal{E}_{min} , then \mathcal{E}_{min} is set to E_{min} . A new random \mathcal{PE} is generated and the outer **while** loop repeats.

3.1.1 Greedy heuristic scheduling algorithm

The greedy heuristic scheduling algorithm (GHSA) performs the task scheduling (Step 3 of 4EMM) for G-Min, G-Deadline, G-Max, and Max-Min. The major difference among the four greedy heuristics is how these heuristics schedule T to \mathcal{PE} . Algorithm 2 shows the pseudo-code for GHSA. GHSA takes as input an ETC matrix, $\mathcal{PE}_p, d_i \forall t_i \in T$, and \mathcal{PE} . The output of GHSA is the T to \mathcal{PE} mapping, E_{sol} , and M . GHSA starts at the first element of the ETC matrix and assigns the task to the most suitable PE. Because a t_i to PE_j mapping must adhere to the d_i constraint, at Line 5, the GHSA heuristic must set PE_j to the minimum DVS level, DVS_1 (Table 2). The DVS_k is incrementally increased until d_i is violated. If the task does not meet the deadline when running at the highest DVS level (DVS_4) then GHSA attempts to assign t_i to the next PE in the ETC matrix. If GHSA fails to schedule t_i to any of the PEs, then the deadline constraint cannot be satisfied and a flag, d_{flag} , is set (Line 9) to indicate there does not exist any feasible solution. When t_i is successfully assigned to a PE, we must take into account the run-time of t_i and the energy consumed by PE_j . In Line 6, $ETC(ij)$ is added to m_j and in Line 7, $EEC(ij)$ is added to E_{sol} . If a feasible solution is obtained, we must calculate the energy consumed, E_{sol} , to process the t_i to \mathcal{PE} mapping. Note that the energy consumed during idle time is accounted for at Line 15. That is,

$$E_{idle} = p_j \cdot t_{idle} \cdot k_{idle}, \tag{15}$$

where t_{idle} is the difference between M and m_j . k_{idle} is a scalar relating the instantaneous power of a PE under load to an idle PE.

Table 2 Power scalars for each DVS level

| DVS level | Speed | Power scalar |
|-----------|-------|--------------|
| 1 | 70% | 0.3430 |
| 2 | 80% | 0.5120 |
| 3 | 90% | 0.7290 |
| 4 | 100% | 1 |

3.1.2 G-Min

The G-Min heuristic (Algorithm 3) schedules the tasks with the shortest execution times first. The motivation behind scheduling the shortest tasks first is to induce slack in the schedule. This slack allows the subsequent tasks with longer execution times to be scheduled without violating the deadline constraints. G-Min receives an ETC matrix as input and outputs the *rearranged* ETC matrix, EEC matrix, and *I-ETC*. Let R be a row in the ETC matrix and C_i be the i th column in the ETC matrix. Note

Input: $ETC, \mathcal{PE}_p, d_i \forall t_i \in T$, and \mathcal{PE}
Output: T to \mathcal{PE} mapping, E_{sol}, M

```

1 foreach  $t_i \in T$  do
2   foreach  $PE_j \in \mathcal{PE}$  do
3     for  $DVS_k = 1$  to 4 do
4       if  $t_{ijk} + m_j \leq d_i$  then
5         Assign  $t_i$  to  $PE_j$  at  $DVS_k$ ;
6          $m_j \leftarrow m_j + ETC(ij)$ ;
7          $E_{sol} \leftarrow E_{sol} + EEC(ij)$ ;
8       end
9     end
10    if  $t_i$  not assigned then
11       $d_{flag} \leftarrow 1$ ;
12      EXIT;
13    end
14  end
15 end
16 foreach  $PE_j \in \mathcal{PE}$  do
17    $E_{sol} \leftarrow E_{sol} + E_{idle}$ ;
18 end

```

Algorithm 2: Greedy heuristic scheduling algorithm (GHSA)

Input: ETC
Output: $ETC, EEC, I-ETC$

```

1 foreach  $R \in ETC$  do
2   Sort  $R$  in ascending order;
3   Sort corresponding row in  $I-ETC$ ;
4 end
5  $\forall R \in ETC$ , swap  $R$  such that  $C_1$  is in ascending order;
6 Apply same changes to  $I-ETC$ ;
7 INVOKE GHSA;

```

Algorithm 3: G-Min

Input: *ETC*
Output: *ETC, EEC, I-ETC*
1 **foreach** $R \in ETC$ **do**
2 Sort R in ascending order according to each t_i 's d_i ;
3 Sort corresponding row in *I-ETC*;
4 **end**
5 $\forall R \in ETC$, swap R such that C_1 is in ascending order;
6 Apply same changes to *I-ETC*;
7 INVOKE GHSA;

Algorithm 4: G-Deadline

Fig. 3 ETC matrix *rearranged* for G-Deadline

| Deadlines | |
|-----------|----|
| t_1 | 13 |
| t_2 | 15 |
| t_3 | 10 |

(a)
Deadlines

| | | |
|----|----|----|
| 8 | 7 | 10 |
| 10 | 9 | 5 |
| 6 | 12 | 7 |

(b)

| | | |
|---|---|----|
| 7 | 8 | 10 |
| 5 | 9 | 10 |
| 6 | 7 | 12 |

(c)

| | | |
|---|---|----|
| 6 | 7 | 12 |
| 7 | 8 | 10 |
| 5 | 9 | 10 |

(d)

ETC Matrix

that G-Min, G-Deadline, G-Max, and MaxMin all have the same inputs and outputs. Figure 2 illustrates the process of *rearranging* the ETC matrix. G-Min *rearranges* the ETC matrix exactly as described in Sect. 3.1.3. After being *rearranged*, the ETC matrix is sent to GHSA to be evaluated.

3.1.3 G-Deadline

One of the major differences between G-Deadline and G-Min is in the task scheduling (Step 3 in 4EMM). In G-Deadline (Algorithm 4), the tasks with the most urgent deadlines are scheduled first. Because tasks are scheduled based on urgency, the tasks that are scheduled later would have a better chance of being scheduled. Figure 3 shows how the ETC matrix is *rearranged*. As seen in Fig. 3(c), the rows are sorted in ascending order. In Fig. 3(d), the rows are swapped such that the execution times in the first column in the ETC matrix are arranged in ascending order based on the task's deadline. After G-Deadline *rearranges* the ETC matrix, GHSA is invoked.

3.1.4 G-Max

In G-Max (Algorithm 5), the tasks with the longest execution times are scheduled first. When the tasks with the longest execution times are scheduled first, only the

Input: *ETC*
Output: *ETC, EEC, I-ETC*

- 1 **foreach** $R \in ETC$ **do**
- 2 Sort R in ascending order;
- 3 Sort corresponding row in *I-ETC*;
- 4 **end**
- 5 $\forall R \in ETC$, swap R such that C_1 is in descending order;
- 6 Apply same changes to *I-ETC*;
- 7 INVOKE GHSA;

Algorithm 5: G-Max

Input: *ETC*
Output: *ETC, EEC, I-ETC*

- 1 **foreach** $R \in ETC$ **do**
- 2 Sort R in descending order;
- 3 Sort corresponding row in *I-ETC*;
- 4 **end**
- 5 $\forall R \in ETC$, swap R such that C_1 is in ascending order;
- 6 Apply same changes to *I-ETC*;
- 7 INVOKE GHSA;

Algorithm 6: MaxMin

Fig. 4 ETC matrix *rearranged* for G-Max

| | | | | | | | | |
|-----|----|----|---|-----|----|---|---|-----|
| 8 | 7 | 10 | 7 | 8 | 10 | 7 | 8 | 10 |
| 10 | 9 | 5 | 5 | 9 | 10 | 6 | 7 | 12 |
| 6 | 12 | 7 | 6 | 7 | 12 | 5 | 9 | 10 |
| (a) | | | | (b) | | | | (c) |

ETC Matrix

tasks with the shortest execution times remain. Because these tasks have the shortest execution times, GHSA can more easily scheduled these tasks without violating the deadline constraints. Figure 4 demonstrates the process of *rearranging* the ETC matrix for G-Max. In Fig. 4(b), the rows of the ETC matrix are sorted in ascending order. In Fig. 4(c), the rows are swapped so that the execution times in the first column are arranged in descending order.

3.1.5 MaxMin

During the initial phase of MaxMin (Algorithm 6), tasks are scheduled to the least efficient PEs. The major motivation behind MaxMin to allow there to be slack in the schedules of the most efficient PEs late in the scheduling process. The subsequent tasks can be executed on the most efficient PEs. Figure 5 shows the process of *rearranging* the ETC matrix. In Fig. 5(b), the rows of the ETC matrix are sorted in

Fig. 5 ETC matrix *rearranged* for MaxMin

| | | | | | | | | |
|-----|----|----|-----|---|---|-----|---|---|
| 8 | 7 | 11 | 11 | 8 | 7 | 10 | 9 | 5 |
| 10 | 9 | 5 | 10 | 9 | 5 | 11 | 8 | 7 |
| 6 | 12 | 7 | 12 | 7 | 6 | 12 | 7 | 6 |
| (a) | | | (b) | | | (c) | | |

ETC Matrix

Input: $ETC, \mathcal{PE}_p, d_i \forall t_i \in T$, and \mathcal{PE}

Output: T to \mathcal{PE} mapping, E_{sol}, M

```

1 foreach  $t_i \in T$  do
2   | Calculate  $TS_i$ ;
3 end
4 Sort  $TS$  in descending order;
5 foreach  $t_i \in TS$  do
6   | foreach  $PE_j \in \mathcal{PE}$  do
7     | Calculate  $PS_{ij}$ ;
8   end
9    $j \leftarrow \arg_j \min(PS_{ij})$ ;
10  for  $DVS_k = 1$  to 4 do
11    | if  $t_{ijk} + m_j \leq d_i$  then
12      | Assign  $t_i$  to  $PE_j$  at  $DVS_k$ ;
13      |  $m_j \leftarrow m_j + ETC(ij)$ ;
14      |  $E_{sol} \leftarrow E_{sol} + EEC(ij)$ ;
15    end
16  end
17  if  $t_i$  not assigned then
18    |  $d_{flag} \leftarrow 1$ ;
19    | EXIT;
20  end
21 end
22 foreach  $PE_j \in \mathcal{PE}$  do
23   |  $E_j \leftarrow E_j + E_{idle}$ ;
24 end

```

Algorithm 7: ObFun

descending order. In Fig. 5(c), the rows are swapped so that the execution times in the first column are arranged in descending order.

3.1.6 ObFun

ObFun is a greedy heuristic that uses two objective functions to determine task to PE mappings. The pseudo-code for ObFun is presented in Algorithm 7. ObFun takes as input an ETC matrix, $\mathcal{PE}_p, d_i \forall t_i \in T$, and \mathcal{PE} . The output of ObFun is T to \mathcal{PE} mapping, the energy consumed by this solution, E_{sol} , and M .

Table 3 Parameters used in TaskSelect and PE Select

| Parameters | |
|------------|-----------|
| α_1 | 0.520656 |
| α_2 | 0.381958 |
| α_3 | 0.0431519 |
| α_4 | 0.160583 |
| α_5 | 0.522339 |
| α_6 | 0.696564 |
| β_1 | 0.0970764 |
| β_2 | 0.400818 |
| β_3 | 0.773407 |

In Line 2, ObFun generates the *TaskSelect* array (*TS*). Every t_i has an entry in *TS* that is based on the following:

$$TS_i = \alpha_1(T_{2,i} - T_{1,i}) + \alpha_2(P_{2,j} - P_{1,j}) + \alpha_3 \frac{T_{1,i} + T_{2,i}}{\sum_{k=1}^{\text{tasks}} (T_{1,k} + T_{2,k})} + \alpha_4 + \alpha_5 + \alpha_6, \quad (16)$$

where $T_{1,i}$ denotes the minimum estimated completion time of t_i . $T_{2,i}$ represents the second shortest estimated completion time of t_i . $P_{1,j}$ and $P_{2,j}$ are the first and second most power-efficient PEs for task t_i respectively. α_{1-3} are weight parameters and α_{4-6} are values added to *TS* if the following conditions are met.

- α_4 is added if the PE with the shortest execution time for t_i is also the most power-efficient.
- α_5 is added if the PE with the shortest execution time for t_i and the PE that is the second most power-efficient are the same, or *vice-versa*.
- α_6 is added if the PE with the second shortest execution time for t_i and the PE that is second most power-efficient are the same.

The values of these parameters are recorded in Table 3.

In Line 4, *TS* is sorted in descending order to allow ObFun to schedule the most appropriate tasks (according to the objective function) first. In Line 7, the most suitable PE for each task is determined and placed in the *PE Select* array, *PS*. Each PE is given a value for every task from the following objective function:

$$PS = \beta_1 T_{1,PE_j,t_i} + \beta_2 P_{1,PE_j,t_i} + \beta_3 load(PE_j), \quad (17)$$

where T_{1,PE_j,t_i} is the execution time of t_i on processor PE_j , P_{1,PE_j,t_i} is the instantaneous power consumption of processor PE_j when executing task t_i , and $load(PE_j)$ is a value added when certain conditions are met. The value of $load(PE_j)$ is zero if t_i satisfies d_i when assigned to PE_j . If t_i does not satisfy d_i , then $load(PE_j)$ equals $m_j - d_i$. Following the above, t_i is assigned to the PE with the lowest *PS* value. In Line 12, Obfun determines the lowest DVS_k that PE_j can be set to before scheduling t_i to PE_j . After t_i is schedules, the executing time of t_i and the energy consumed by

Input: $ETC, \mathcal{PE}_p, d_i \forall t_i \in T$, and \mathcal{PE}
Output: T to \mathcal{PE}, E_{sol}, M

```

1  foreach  $t_i \in T$  do
2  |   foreach  $PE_j \in \mathcal{PE}$  do
3  |   |   for  $DVS_k = 1$  to 4 do
4  |   |   |   Calculate  $U_{tijk}$ ;
5  |   |   end
6  |   end
7  end
8  foreach  $t_i \in T$  do
9  |    $j, k \leftarrow \arg_{j,k} \max(U_{tijk});$ 
10 |   if  $t_{ijk} + m_j \leq d_i$  then
11 |   |   Assign  $t_i$  to  $PE_j$  at  $DVS_k$ ;
12 |   |    $m_j \leftarrow m_j + ETC(ij);$ 
13 |   |    $E_{sol} \leftarrow E_{sol} + EEC(ij);$ 
14 |   else
15 |   |    $U_{tijk} \leftarrow 0;$ 
16 |   end
17 |   if  $t_i$  not assigned then
18 |   |    $d_{flag} \leftarrow 1;$ 
19 |   |   EXIT;
20 |   end
21 end
22 foreach  $PE_j \in \mathcal{PE}$  do
23 |    $E_{sol} \leftarrow E_{sol} + E_{idle};$ 
24 end

```

Algorithm 8: UtFun

PE_j must be recorded. In Line 13, $ETC(ij)$ is added to m_j , and in Line 14, $EEC(ij)$ is added to E_{sol} . If t_i can not meet d_i when PE_j is running at the highest DVS level (DVS_4), then a flag is set (Line 16) to indicate a feasible solution does not exist. If a feasible solution is found, then the total energy consumption of the solution is calculated in a manner analogous to GHSA.

3.1.7 UtFun

The UtFun heuristic uses a utility function to determine task to PE assignments. Utility functions are often used in economics to measure the relative benefits from various goods and services [28]. In UtFun, the utility function calculates the benefit gained from each task to PE assignment.

Algorithm 8 displays the pseudo-code for UtFun. The inputs to UtFun are an ETC matrix, $\mathcal{PE}_p, d_i \forall t_i \in T$, and \mathcal{PE} . The outputs of UtFun are the T to \mathcal{PE} mapping, E_{sol} , and M . UtFun enters a loop where the utility of each task is calculated for each PE and DVS level (Line 4). The utility is a function of the PE’s speed and the

execution time of the task. The speed and utility can be represented by:

$$S(v) = \frac{k_1 \cdot (v - v_t)_2}{v}, \quad (18)$$

$$Utility = S^p \cdot T^q, \quad (p, q > 0), \quad (19)$$

where p determines the relative importance of the speed, and q determines the relative significance of the execution time of the task. In Line 11, the task is assigned to the PE and DVS level that yield the highest utility. Assigning t_i to the PE_j with the highest utility does not guarantee that the deadline constraint will be satisfied. Violating the deadline will cause the utility for this specific t_i - PE_k pair to be set to zero (Line 15). UtFun identifies the PE-DVS level pair with the next highest utility and assigns the task to this PE. If a valid task assignment has occurred, then the execution time of t_i and the energy consumed by PE_j must be taken into account. $ETC(ij)$ is added to m_j (Line 12) and $EEC(ij)$ is added to E_{sol} . If the task cannot be assigned to any of the PEs without breaking the deadline, then d_{flag} is set to indicate that a feasible solution does not exist. If a feasible solution is found, then the total energy consumption is calculated in a manner analogous to GHSA and ObFun.

3.2 Genetic algorithms

Genetic algorithms are a type of evolutionary algorithm that are modeled after biological evolution. At the beginning of a genetic algorithm, an initial population of solutions is randomly generated. After solution initialization, there are four steps in a genetic algorithm that repeat until a halting condition is reached.

1. **Evaluation:** The costs of every solution are calculated.
2. **Ranking:** Each solution is ranked based on their costs.
3. **Reproduction:** The highest ranked solutions are modified via *crossover* and *mutation*.
4. **Replacement:** The lowest ranked solutions are replaced by the reproduced solutions.

Every time the algorithm completes the above mentioned four steps, a *generation* has completed. After the evaluation step, the genetic algorithm checks if the halting condition has been reached.

Each solution in a genetic algorithm is represented by a *chromosome*. Figure 6 gives an example of a chromosome. Every element of the chromosome represents a task and contains an integer specifying the PE to which the task is assigned. For instance, we can observe that PE_1 is assigned t_1 and t_{i-1} , PE_2 is assigned t_3 , and PE_3 is assigned t_1 . A natural curiosity at this point would be which of the two tasks, t_2 or t_{i-1} , is first executed on PE_1 . This is Step 2 in 4EMM, which is the resource matching step. The actual scheduling, that is which task executes first on a given PE, is part of Step 3 in 4EMM. Step 3 is achieved by invoking GenAlg or GenAlg-DVS. (GenAlg and GenAlg-DVS will be explained in the subsequent text.)

In this paper, solution ranking is determined by solution *domination* and *Pareto-ranks*. A given solution *dominates* another solution if every objective is better. The

objectives examined in this paper are *makespan* and energy consumption. A solution's *Pareto-rank* is the number of other solutions that do not dominate it. In Fig. 7, each solution is represented by a circle. The solution's *makespan* and energy consumption are indicated by the position on the graph. The *Pareto-rank* of each solution is indicated by the number in each circle. The lowest ranked solution has a *Pareto-rank* of 1 because only one solution (Circle 3) does not have a both a shorter *makespan* and a lower energy consumption. The highest ranked solutions have a *Pareto-rank* of 5 because none of the other five solutions have both a shorter *makespan* and a lower energy consumption.

In genetic algorithms, there are two genetic operators used to maintain genetic diversity. *Crossover* is a genetic operator used to vary the solutions from one generation to the next. GA-A uses a two-point *crossover* technique. Two-point *crossover* selects the same cut points in two parent chromosomes, then swaps the information between the two points, creating two child chromosomes. Figure 8 illustrates two-point *crossover*. In this example, the first cut is made between t_2 and t_3 , and the second cut is made between t_7 and t_8 . Examination of the child chromosomes reveals that the tasks between the two cuts in the parent chromosomes have been swapped.

Mutation is another genetic operator used to maintain genetic diversity in the solutions from one *generation* to the next. During *mutation*, a number of task assignments in a solution will randomly be reassigned to a different PE. This random assignment is performed to avoid local minimas.

Fig. 6 Chromosome

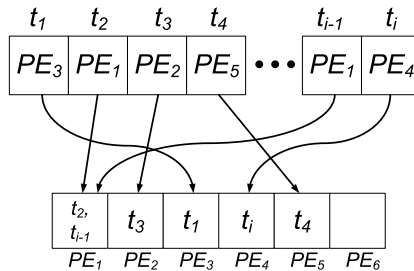


Fig. 7 Pareto-Rank

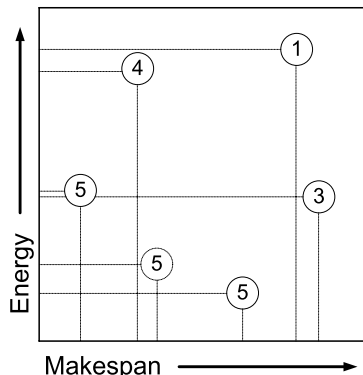
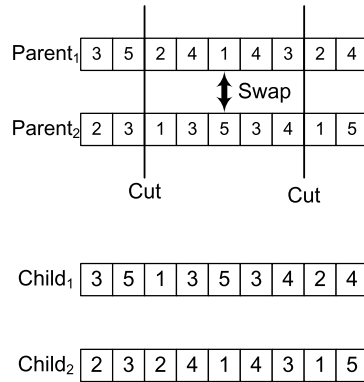


Fig. 8 Crossover



3.2.1 GA-A

The GA-A algorithm is a naturally inspired genetic algorithm that maintains a pool of solutions and follows the four steps of genetic algorithms. GA-A does the resource allocation (Step 1 of 4EMM) and resource matching (Step 2 of 4EMM) for both GenAlg and GenAlg-DVS. Because we want to have a diverse population, an intuitive way to generate solutions would be to consider multiple \mathcal{PE} 's. The pool of solutions is randomly divided into multiple solution clusters, each with a different \mathcal{PE} . Algorithm 9 shows the pseudo-code for GA-A. The inputs to GA-A consist of an ETC matrix, \mathcal{PE}_p , and $d_i \forall t_i \in T$ and outputs the T to \mathcal{PE} mapping, \mathcal{E}_{\min} , and M . In Line 2, GA-A randomly generates initial solutions. GA-A invokes GenAlg or GenAlg-DVS to evaluate the initial solutions in Line 3. GenAlg and GenAlg-DVS will be described in detail later in this section. For the time being, assume GenAlg and GenAlg-DVS are generic solution evaluation procedures. Note that GenAlg or GenAlg-DVS is invoked in Lines 3, 8, and 22. After the initial solutions have been evaluated, GA-A enters a nested **while** loop (Lines 4 and 5) that repeats until the halting condition has been satisfied. After every *generation*, the outer **while** loop (Line 4) calculates the global minimum energy consumption, \mathcal{E}_{\min} , and the inner **while** loop records the local minimum energy consumption, E_{\min} .

In Line 6, every solution is ranked based on solution *domination*. The solution reproduction step within a genetic algorithm can be carried out after the solutions have been ranked. This is achieved by discarding the lowest ranked solutions in C_m , and reproducing high-ranked solutions to take their place. The reproduced solutions are modified via *crossover* or *mutation*, as previously described. A system variable, p_m , determines the probability that a newly reproduced solution will be changed via mutation or crossover.

After the solution reproduction step, GA-A invokes GenAlg or GenAlg-DVS to re-evaluate the solutions. If any of the solutions are more energy-efficient than the previous best, then k_2 is set to zero and E_{\min} is updated. If a more energy efficient solution has not been found in the last $k_{2,\max}$ generations, then the inner **while** loop (Line 5) terminates. Because we want to give GA-A the opportunity to find a better solution, k_1 is set to zero if $E_{\min} < \mathcal{E}_{\min}$. Otherwise, k_1 is incremented. If $k_1 > k_{1,\max}$, then the program terminates.

Input: $ETC, \mathcal{PE}_p, d_i \forall t_i \in T$
Output: T to \mathcal{PE} mapping, \mathcal{E}_{\min}, M

```

1 Generate  $\mathcal{C}$ ;
2 Generate Initial Solutions;
3 INVOKE GenAlg/GenAlg-DVS;
4 while  $k_1 \leq k_{1,\max}$  do
5   while  $k_2 \leq k_{2,\max}$  do
6     Rank Solutions;
7     Carry out Solution Reproduction;
8     INVOKE GenAlg/GenAlg-DVS;
9     if Any  $S_n \in C_m$  Improved then
10      |  $k_2 \leftarrow 0$ ;
11     else
12      | INCREMENT  $k_2$ ;
13     end
14   end
15   if Any  $C_m \in \mathcal{C}$  Improved then
16     |  $k_1 \leftarrow 0$ ;
17   else
18     | INCREMENT  $k_1$ ;
19   end
20   Rank Clusters;
21   Carry out Cluster Reproduction;
22   INVOKE GenAlg/GenAlg-DVS;
23 end

```

Algorithm 9: GA-A

In Line 20, cluster ranking is performed. Clusters are composed of multiple solutions, so each cluster contains many sets of costs. The clusters are ranked with partial domination [12]. Cluster domination, C_{dom} , is represented by a scalar value instead of a Boolean value. Let x and y be clusters and $NIS(x)$ be the set of noninferior solutions in x . If a and b are solutions in C_x , then $DOM(a, b)$ is equal to 1 if a is not dominated by b and 0 otherwise. Mathematically, cluster domination can be represented by:

$$C_{\text{dom}}(x, y) = \max(a \in NIS(x)) \sum_{b \in NIS(y)} DOM(a, b), \tag{20}$$

$$C_{\text{rank}}[x] = \sum_{y \in \mathcal{C} \forall x \neq y} C_{\text{dom}}(x, y). \tag{21}$$

From the above, we can observe that cluster ranking can be obtained by summing the C_{dom} value for each cluster. After the clusters have been ranked, cluster reproduction is carried out in a manner analogous to solution reproduction (Line 21). The lowest ranked clusters are deleted and replaced by newly reproduced clusters. The

Input: $ETC, EEC, d_i \forall t_i \in T, \mathcal{C}$, and T
Output: $E_{sol} \forall S_n \in \mathcal{C}$ and M

```

1  foreach  $C_m \in \mathcal{C}$  do
2  |   foreach  $S_n \in C_m$  do
3  |   |    $E_{S_n} \leftarrow 0$ ;
4  |   |   foreach  $t_i \in T$  do
5  |   |   |   if  $ETC(i,j) + m_j \leq d_i$  then
6  |   |   |   |   Assign  $t_i$  to  $PE_j$ ;
7  |   |   |   |    $m_j \leftarrow m_j + ETC(i,j)$ ;
8  |   |   |   |    $E_{S_n} \leftarrow E_{S_n} + EEC(i,j)$ ;
9  |   |   |   else
10 |   |   |   |    $S_n$  is invalid;
11 |   |   |   end
12 |   |   end
13 |   end
14 end

```

Algorithm 10: GenAlg

reproduced clusters are modified via mutation or crossover. Cluster mutation and crossover are analogous to solution mutation and crossover.

3.2.2 GenAlg

The GenAlg algorithm carries out the task scheduling (Step 3 in 4EMM) and evaluates the solutions (Step 4 in 4EMM) that are produced in GA-A. Algorithm 10 shows the pseudo-code for GenAlg. The inputs for GenAlg are an ETC matrix, $\mathcal{P}\mathcal{E}_p$, $d_i \forall t_i \in T, \mathcal{C}$, and $\mathcal{P}\mathcal{E}$. GenAlg outputs E_{sol} and M . The first two **for** loops (Lines 1 and 2) iterate through every solution in \mathcal{C} . In Line 3, the energy consumption of S_n is set to 0 because no tasks have been scheduled. The next **for** loop (Line 4) iterates through S_n and the tasks are scheduled to the specified PE (Line 6) if the deadline constraint is not violated. The run-time and energy consumption of t_i is added to PE_j 's total run-time (m_j) and E_{S_n} , respectively. If the task cannot be scheduled to PE_j without violating the deadline constraint, then the corresponding solution is invalid. Notice that GenAlg does not make use of the PEs' DVS levels. We wanted to see how the results of a genetic algorithm, which has a global viewpoint of the problem, are effected by DVS techniques.

GenAlg-DVS (Algorithm 11) makes use of the PEs' DVS modules. Because GenAlg-DVS uses DVS levels, we will be able to compare the results to GenAlg and see how using the PE's DVS module affects the energy consumption of the solution. The inputs and outputs of GenAlg-DVS are the same as GenAlg's. The first difference between GenAlg and GenAlg-DVS can be seen in Line 5. Two variables are introduced (Lines 5 and 6) to control the **while** loop at Line 7. Let t_a denote whether or not the task has been assigned, and k be the DVS level. If t_a is equal to 0, then the task has not been scheduled to a PE. If t_i cannot be scheduled to PE_j at

Input: $ETC, EEC, d_i \forall t_i \in T, \mathcal{C},$ and T
Output: $E_{sol} \forall S_n \in \mathcal{C}$ and M

```

1  foreach  $C_m \in \mathcal{C}$  do
2      foreach  $S_n \in C_m$  do
3           $E_{S_n} \leftarrow 0;$ 
4          foreach  $t_i \in T$  do
5               $t_a \leftarrow 0;$ 
6               $k \leftarrow 1;$ 
7              while  $t_a = 0 \ \& \ k \leq 4$  do
8                  if  $t_{ijk} + m_j \leq d_i$  then
9                      Assign  $t_i$  to  $PE_j$  at  $DVS_k$ ;
10                      $t_a \leftarrow 1;$ 
11                      $m_j \leftarrow m_j + t_{ijk};$ 
12                      $E_{S_n} \leftarrow E_{S_n} + EEC(ij);$ 
13                 else
14                      $k \leftarrow k + 1;$ 
15                 end
16             end
17             if  $t_a = 0$  then
18                  $S_n$  is invalid;
19             end
20         end
21     end
22 end

```

Algorithm 11: GenAlg-DVS

DVS_k without violating the deadline constraint, then k is incremented at Line 14. If t_i cannot satisfy the deadline constraint at DVS_4 , then S_n is invalid.

4 Simulations, results, and discussion

All of the heuristics introduced in this paper were implemented in Matlab. Matlab can efficiently perform operations on large matrices [29]. Because our simulations make use of large matrices, using Matlab appeared to be the best choose. The dimensions of the ETC matrix used in our simulation were as large as 100,000 tasks by 16 PEs. Our results were obtained on a 2.4 GHz Core 2 Duo system with 2 GB of main memory running the Windows 7 operating system.

The set of tasks used in this simulation study were obtained from an ETC matrix (explained in the subsequent text). There were two major goals for our simulation study:

1. To compare and analyze the performance of the eight introduced scheduling heuristics.
2. To measure the impact of system parameter variation.

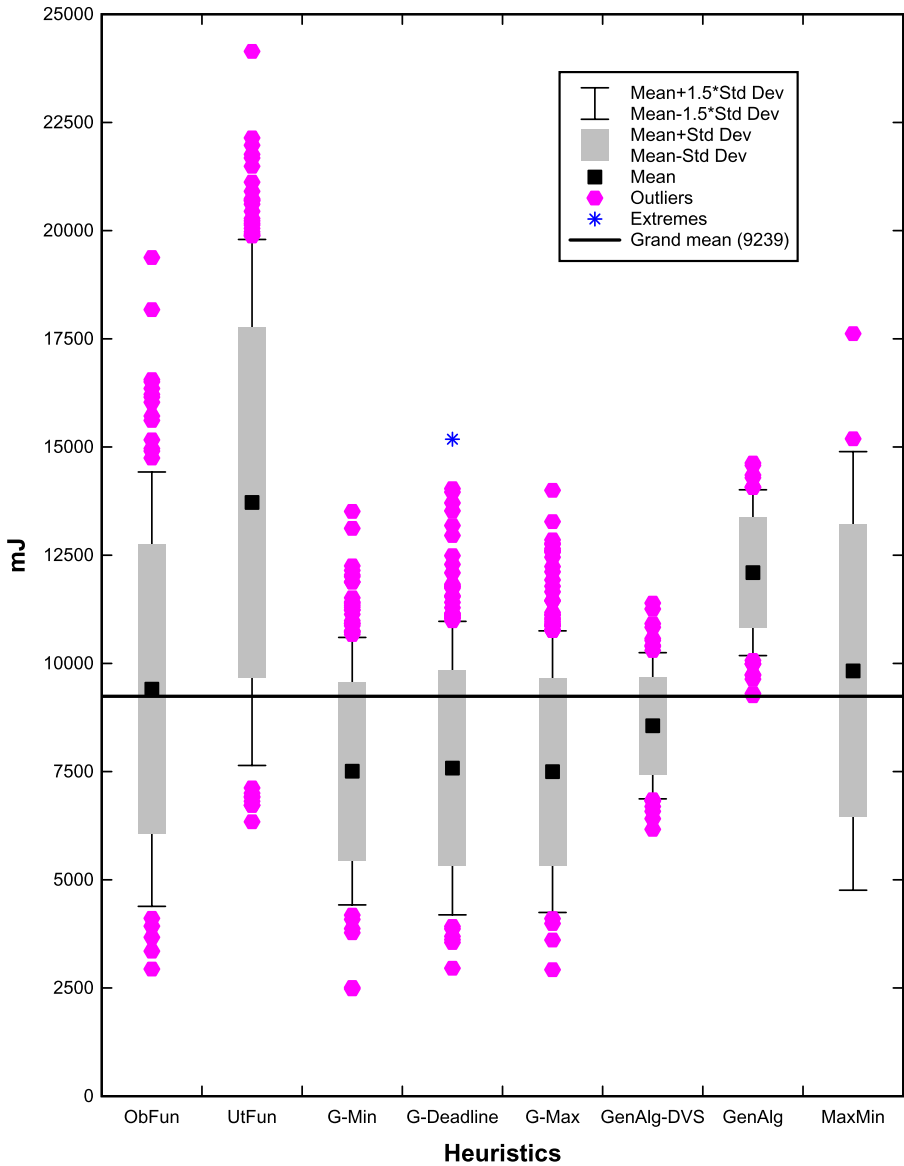
Table 4 Summary of system parameters

| System parameters | | |
|---------------------|---------------------------------|-------------------------------|
| μ_{task} | Average task execution time | 10 |
| V_{task} | Variance in task execution time | {0.1, 0.15, 0.35} |
| V_{PE} | Variance in PE heterogeneity | {0.1, 0.15, 0.35} |
| k_d | Deadline scaling variable | {1, 1.3, 1.8} |
| $ \mathcal{PE} $ | Number of PEs | 16 |
| $ T $ | Number of tasks | {100, 1,000, 10,000, 100,000} |
| DVS_k | Number of DVS Levels | 4 |

4.1 Workload

Based on the number of tasks, the simulation was divided into two parts. For small-sized problems (up to 1,000 tasks), all of the eight proposed heuristics were evaluated. Due to the long run-times of GenAlg and GenAlg-DVS, it became impractical to compute a solution for a problem with more than 1,000 tasks. For large-sized problems (more than 1,000 tasks and up to 100,000 tasks), GenAlg and GenAlg-DVS were not evaluated.

For the workload, we obtained task characteristics from an ETC matrix. An explanation of the generation of our CVB ETC matrix was detailed in Sect. 3.1. The mean task execution time, μ_{task} , was fixed at 10, while the variance in the tasks, V_{task} , and the variance in the PEs, V_{PE} , varied between 0.1 to 0.35. These values were chosen to incorporate variance in our task execution times and are supported in previous studies, such as [21, 24, 30], and are derived from real world applications. The deadline, d_i , of each t_i is based on the ETC matrix and given by (14). To vary the heterogeneity of d_i , the k_d parameter in (14) is varied from 1 and 1.8. For small-size problems, the number of tasks was varied from 100 to 1,000 and the number of PEs was set to 16 [31]. One can choose a large number of PEs; however, studies show that in essence the number of PEs proportionally relates to the number of tasks [32]. Therefore, is one must have 256 PEs to choose from, then they must have at least 500,000 tasks to solve. The dollar cost constraint, D , is obtained by multiplying the number of PEs by the average price per PE. We used an average cost of \$10 per PE, which gives us a total dollar cost constraint of \$160. The dollar cost constraint is only a number which we assigned. We could replace D with a different number and the solution would not be affected. To create a PE pool, \mathcal{PE}_p , with high heterogeneity, some PEs execute tasks faster than other PEs. Because faster PEs generally cost more than slower PEs, the costs of the PEs in \mathcal{PE}_p proportionally relate to their speeds. The number of DVS levels was set to 4. We admit that having larger numbers of DVS levels can produce refined solutions. However, the general characteristics of the algorithms will have no bearing on larger or smaller numbers of DVS levels [20, 33–35]. For large-size problems, the number of tasks varied from 10,000 to 100,000. The rest of the parameters were kept the same as those for the small-size problems. To facilitate readability, all of the above system parameters are summarized in Table 4.



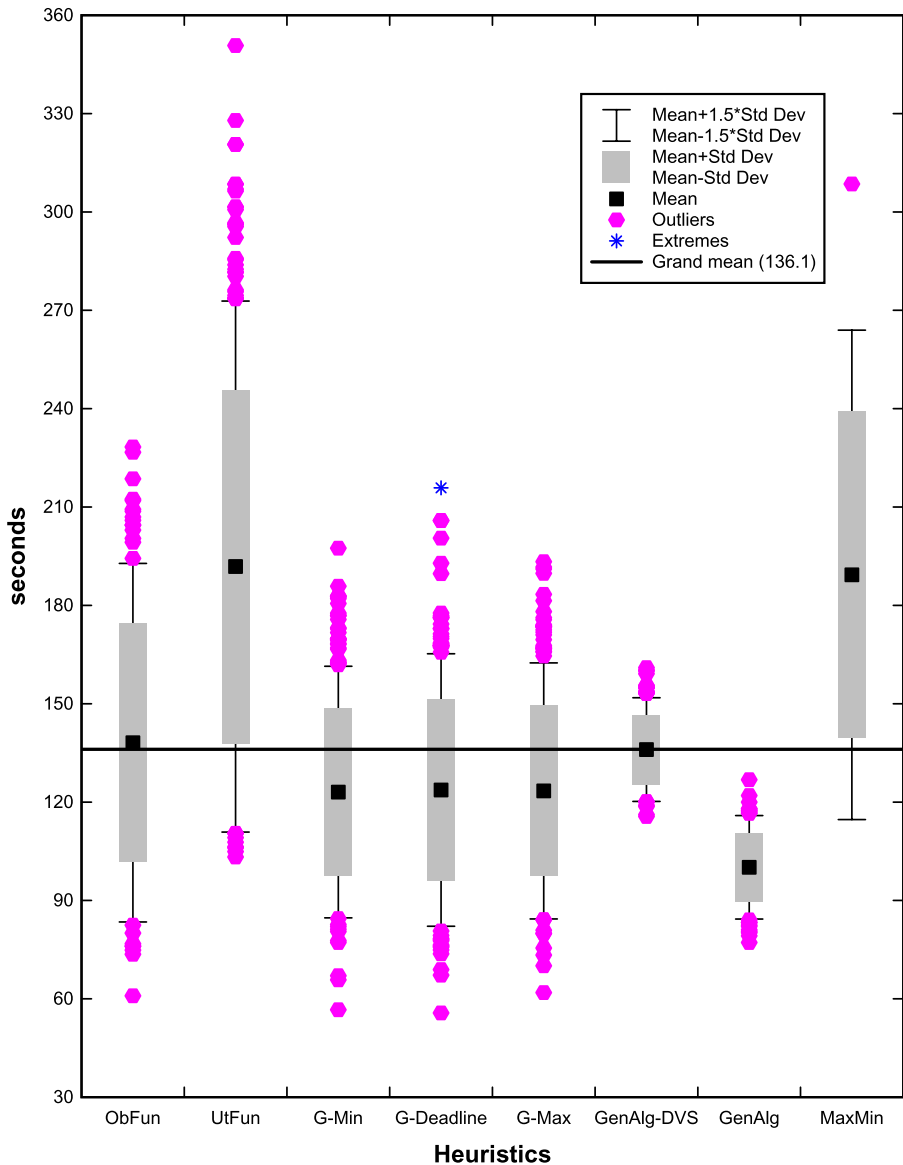
(a) Energy consumption for 100 tasks

Fig. 9 100 task problem-size

4.2 Comparative results

4.2.1 Small-size problems

The simulation results for the small-size problems are shown in Figs. 9, 10, 11. These figures show the average energy consumption and *makespan* of the eight proposed

(b) *Makespan* for 100 tasks**Fig. 9** (Continued)

heuristics. To thoroughly benchmark our heuristics, we considerably varied the simulation system parameters. The V_{task} , V_{PE} , and k_d parameters each have three possible values as observed in Table 6. That means that there will be 3^3 combinations, which gives us a total of 27 sets of parameters. This represents every combination of the system parameters listed in Table 4. To gain confidence in our results, the simulations

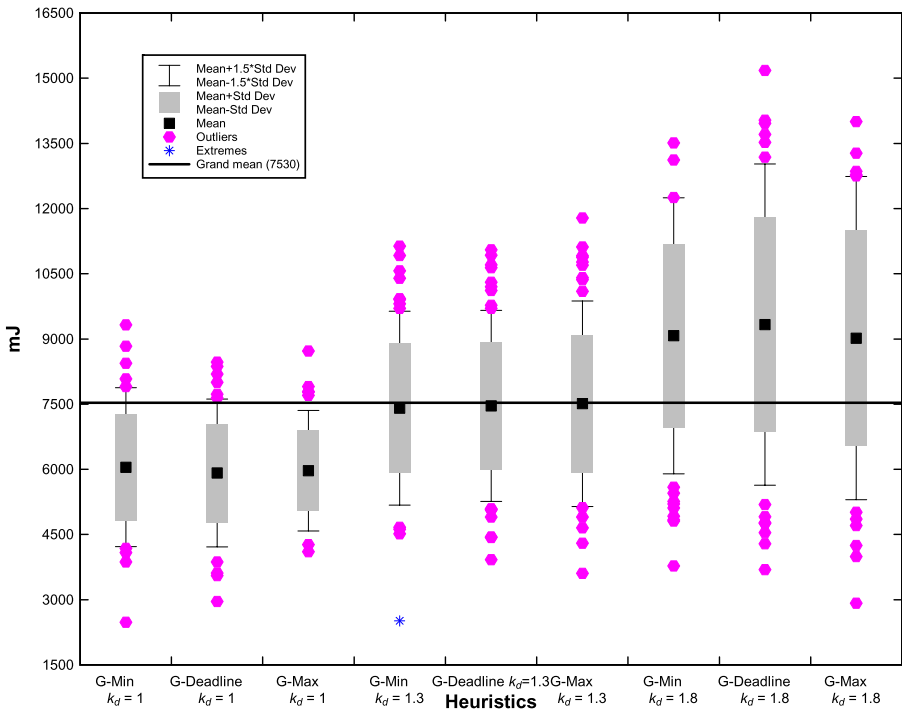
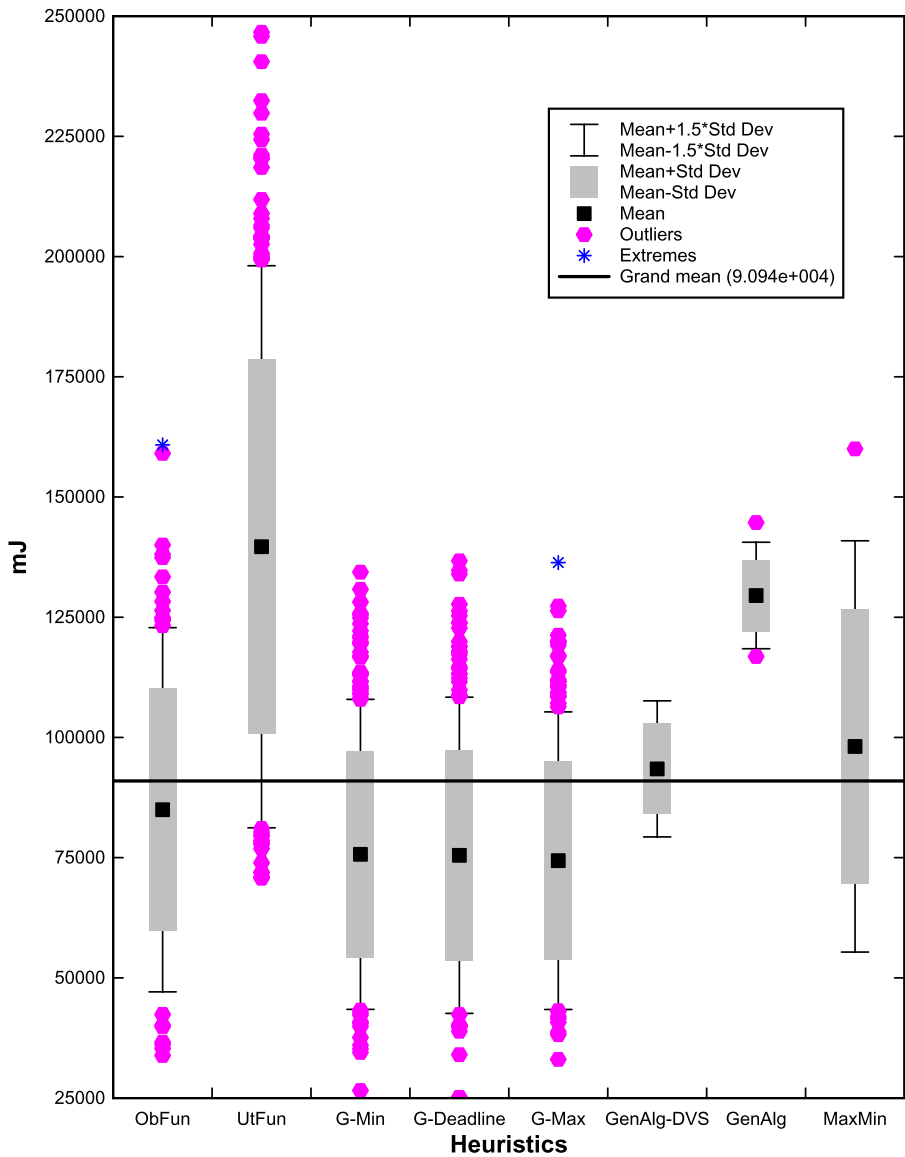


Fig. 10 Energy consumption for 100 tasks with varied k_d

were run ten times for each set of parameters. That is a total of 270 simulations per heuristic.

There is a great deal of information that can be gathered from the plots. The gray box is the range that represents ± 1 times the standard deviation. The mean is represented by a black box in the middle of the gray box. The whiskers extend to ± 1 times the standard deviation. The bold line that spans the entire plot is the grand mean. The outliers and extremes are denoted by circles and asterisks, respectively. In the subsequent text, we will discuss the results for 100, 1,000, 10,000, and 100,000 tasks.

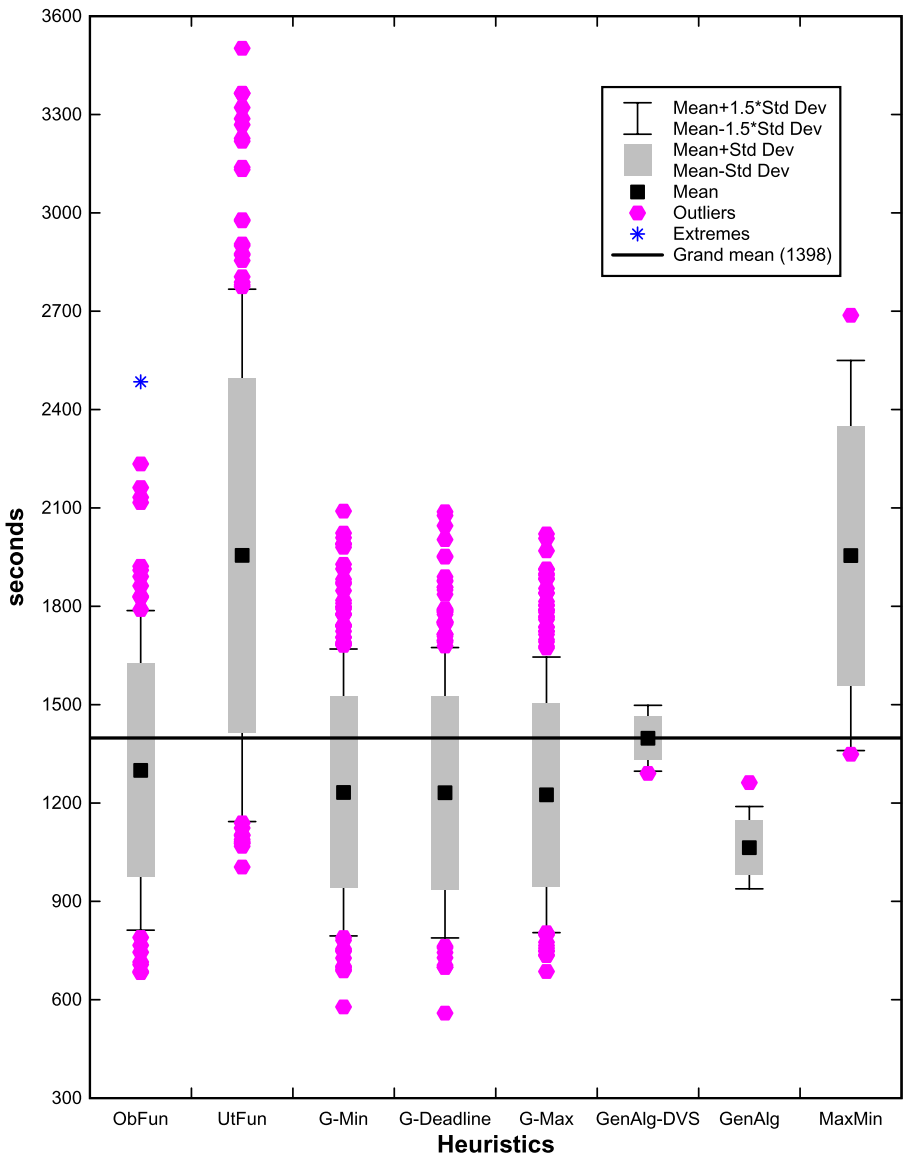
100 tasks The plot in Fig. 9(a) shows that among the eight heuristics, G-Min, G-Deadline, and G-Max consumed the least amount of energy when scheduling 100 tasks. These three heuristics produced very similar results. Based on the mean value, G-Max consumed the least energy (1.09% less than G-Deadline). G-Max schedules the tasks with the longest execution times first. Because the later tasks have the shortest execution times, there is enough slack in the schedule to fit these tasks. However, based on the range of the minimum execution times, G-Min performed better than the other seven heuristics. G-Min had a lower minimum and maximum energy consumption than G-Max. Because G-Min schedules the tasks with the shortest execution times first, there is slack in the schedule for the subsequent tasks, which have the longest execution times. These results show that the motivation behind both G-Min



(a) Energy consumption for 1,000 tasks

Fig. 11 1,000 task problem-size

and G-Max are effective in producing high quality solutions. UtFun had the worst performance among the eight heuristics. The performance was the worst because the utility function used in UtFun favors the PEs with mid-ranged energy consumption. Such a favoritism may not assign tasks in an energy efficient manner. Note that G-Min, G-Deadline, G-Max, and GenAlg-DVS all produced a mean energy consumption lower than the grand mean.



(b) Makespan for 1,000 tasks

Fig. 11 (Continued)

The makespan obtained by each heuristic can be seen in Fig. 9(b). GenAlg exhibited the lowest average makespan. Because GenAlg does not make use of DVS techniques, the makespan was shorter than any other heuristic. MaxMin had the longest makespan. Initially, MaxMin assigns tasks to the least efficient PEs. This is done to allow slack in the schedules of the more efficient PEs. For the heuristics that make use of DVS, G-Min, G-Deadline, and G-Max obtained extremely compa-

Table 5 Average run-time in seconds

| No. of tasks | 100 | 1,000 | 10,000 | 100,000 |
|--------------|--------------|--------------|--------|---------|
| ObFun | $3.87E^{-3}$ | $2.90E^{-2}$ | 0.439 | 21.2 |
| UtFun | $9.44E^{-3}$ | $7.55E^{-2}$ | 0.707 | 7.18 |
| G-Min | $3.02E^{-3}$ | $2.43E^{-2}$ | 0.233 | 2.38 |
| G-Deadline | $2.90E^{-3}$ | $2.38E^{-2}$ | 0.228 | 2.34 |
| G-Max | $3.00E^{-3}$ | $2.50E^{-2}$ | 0.235 | 2.47 |
| MaxMin | $5.07E^{-3}$ | $4.12E^{-2}$ | 0.404 | 4.04 |
| GenAlg | 156 | 2313 | DNE | DNE |
| GenAlg-DVS | 185 | 2426 | DNE | DNE |

rable *makespans* (0.51% difference between the mean of the three). G-Min had the lowest mean *makespan*. G-Deadline had a slightly lower absolute minimum energy consumption than G-Min and G-Max, but also had a significantly higher maximum *makespan*.

G-Min and G-Max take opposite approaches when scheduling tasks. G-Min assigns the tasks with the shortest execution times first, while G-Max assigns the tasks with the longest execution times first. Because G-Deadline assigns tasks based on their deadlines, it does not factor in their execution times. This means that G-Deadline should often times give results that are in between the results of G-Min and G-Max. When looking at specific cases, we can observe that certain heuristics perform better with different simulation system parameters. We observed that with a loose deadline ($k_d = 1.8$), G-Max performed well. With a mid-ranged deadline ($k_d = 1.3$), G-Min produced better results. When the deadline got tight ($k_d = 1$), G-Deadline outperformed G-Min and G-Max. Figure 10 depicts the above results.

1,000 tasks Figure 11(a) shows the energy consumption for 1,000 task problems. Again, G-Min, G-Deadline, and G-Max consumed the least energy. Observer that G-Max had the lowest mean energy consumption (1.71% less than G-Min) and G-Deadline had the lowest minimum energy consumption. The above results validate what we discovered in our 100 task simulations. Our results show that G-Deadline has a larger range of minimum execution times than G-Min or G-Max. G-Deadline had a lower global minimum and a higher global maximum. We can see that UtFun consumed the most energy of any of the heuristics. There are a few key differences between our 100 and 1,000 task simulations. First, we can see that ObFun demonstrated a much better mean energy consumption. ObFun's mean energy consumption is now lower than the grand mean. We can also observe that GenAlg-DVS performed worse in the simulations with 1,000 tasks than it did in the 100 task simulations. GenAlg-DVS is a genetic algorithm that depends on chance to produce a quality solution. As the number of tasks increase, the number of possible solutions exponentially increases. This means that as the number of tasks increase, the execution time needed by GenAlg-DVS to render a good solution is greatly increased. Table 5 shows that GenAlg-DVS has an execution time of over 40 minutes when solving a 1,000 task problem. Allowing GenAlg-DVS to execute any longer would be impractical.

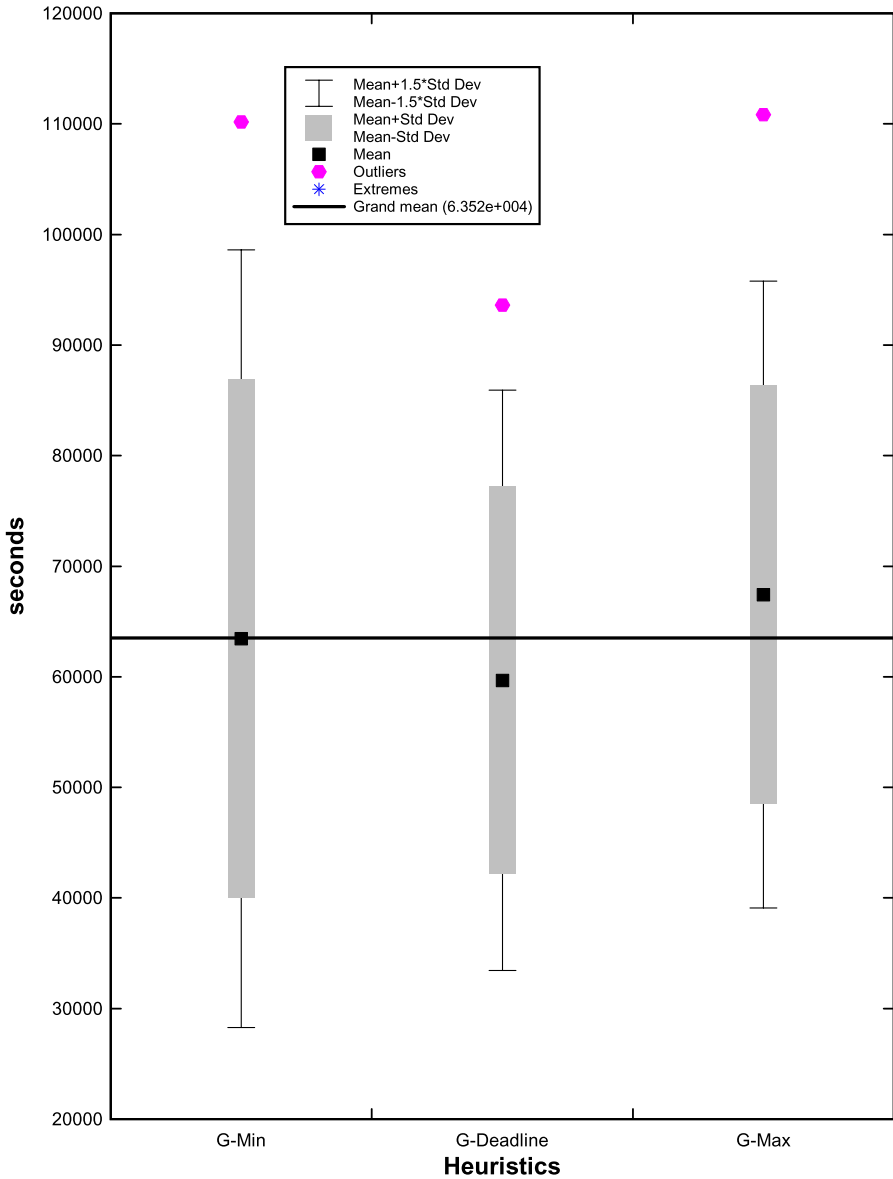


Fig. 12 Energy consumption for 1,000 tasks with $k_d = 1.8$ and high heterogeneity ($V_{task} = V_{PE} = 0.35$)

Figure 11(b) depicts the *makespan* of the eight heuristics. Notice that GenAlg obtained the lowest *makespan* of any heuristic. Because GenAlg executes the PEs at the fastest speed available, the *makespan* is minimized. The problem with this approach is that only the *makespan* is minimized. Figure 11(a) shows that GenAlg exhibits a poor mean energy consumption. When analyzing *makespan*, GenAlg-DVS must be compared to the rest of the heuristics to give us a fair comparison. G-Min, G-

Deadline, and G-Max all had a mean *makespan* within 0.57% of each other, so there is not one heuristics that is significantly better than the others. If we look at values from individual sets of parameters, then there may be some situations where a certain heuristic performs better than the others. When k_d was set to 1.8 and there was high heterogeneity in the ETC matrix ($V_{\text{task}} = V_{PE} = 0.35$), G-Max had a mean *makespan* 11.42% higher than G-Deadline. These results are depicted in Fig. 12. When there is a high degree of heterogeneity in the ETC matrix, there are more tasks with longer execution times. Because there is an increased number of tasks with longer execution times, G-Max has less slack towards the end of the scheduling. This leaves less slack before the deadline to schedule the shorter tasks and in turn increases the *makespan*. When k_d was 1.3 and there was medium heterogeneity ($V_{\text{task}} = 0.35$ and $V_{PE} = 0.1$), ObFun produced a *makespan* 13.14% lower than G-Min.

4.2.2 Large-size problems

10,000 task problem size As mentioned previously, GenAlg and GenAlg-DVS are not included in the simulations for large-sized problems due to their slow termination times. Figure 14(a) shows that there are four heuristics with highly comparable results, namely G-Min, G-Deadline, G-Max, and ObFun. ObFun obtained a mean energy consumption only 3.48% greater than the G-Min. We also can observe that as the problem size increases, ObFun performs better. In certain cases, ObFun obtained the lowest mean energy consumption. Figure 13 illustrates the mean energy consumption when V_{PE} is set to 0.1 and V_{task} is set to 0.35. In the above case, ObFun had a mean energy consumption 8.65% lower than any other heuristic. When there is high task heterogeneity, the objective function used in ObFun (16), is especially effective. Equation (16) considers the tasks with the first and second shortest execution times. When the heterogeneity of the tasks is high, it is important to inspect more than one task during the task scheduling process. Because ObFun considers multiple tasks with its objective function, ObFun produced better results in this case.

The plot in Fig. 14(b) shows that for a 10,000 task problem, ObFun identifies the lowest mean *makespan*. The *TaskSelect* and *PE Select* objective functions introduced in ObFun factor in the loads of each PE when scheduling tasks. This prevents ObFun from scheduling a majority of the tasks to a few (most efficient) PEs. This induced a scheduling slack for the later tasks. When there are more tasks in the problem, it becomes critical that tasks are more evenly distributed among the PEs. Again, UtFun had the highest mean energy consumption. This shows that the utility function used in UtFun (19) does not make good decisions when selecting PE-task pairs. UtFun tends to assign tasks to PEs with mid-range efficiency. Our results show that heuristics that initially assign tasks to the most efficient PEs exhibit lower better results.

100,000 tasks Figures 15(a) and (b) details the energy consumption and *makespan* of the eight heuristics with a 100,000 task problem. ObFun had the lowest mean energy consumption and was 55.18% smaller than the next lowest solution. The lowest mean *makespan* was also achieved by ObFun (41.56% lower than any other heuristic). Because the objective functions implemented in ObFun examine the effects of multiple tasks and multiple PEs before selecting a task-PE pair, ObFun performs extremely well in large-sized problems. We can see that UtFun had the largest mean

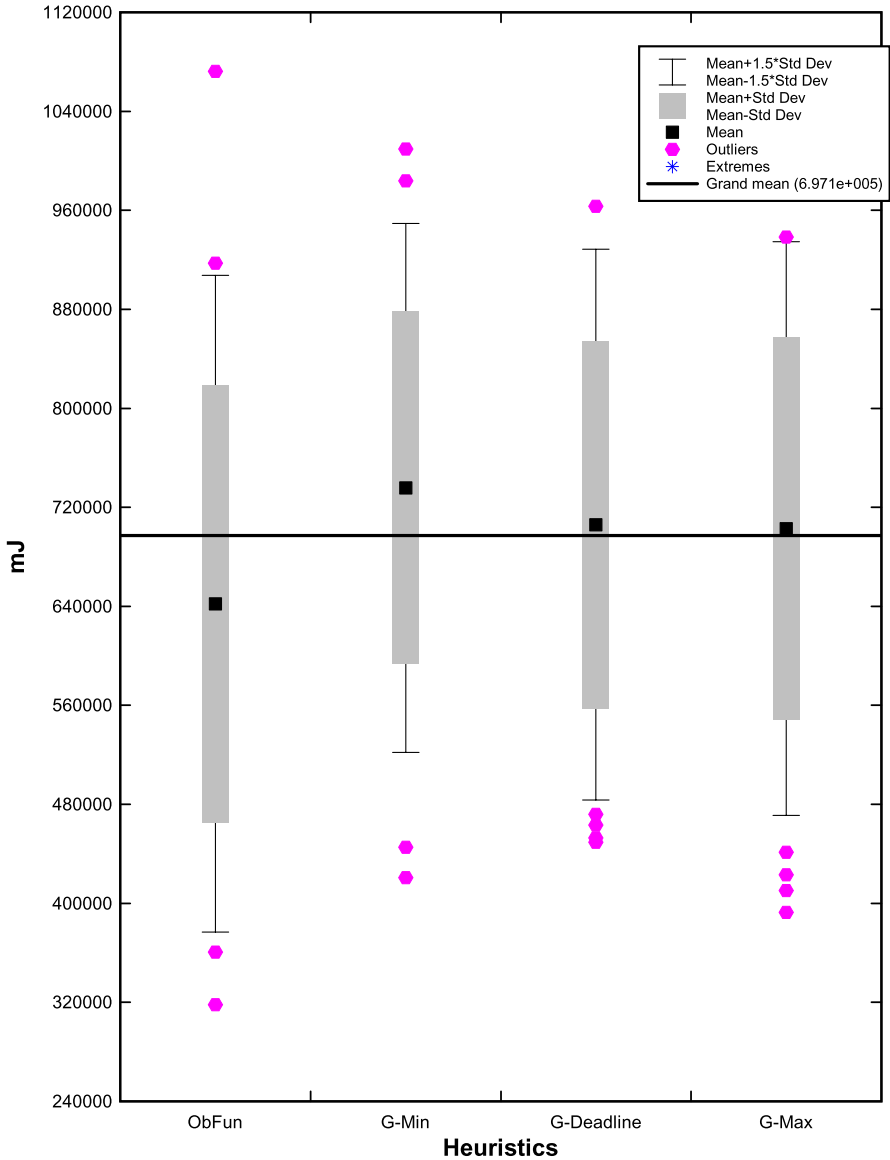
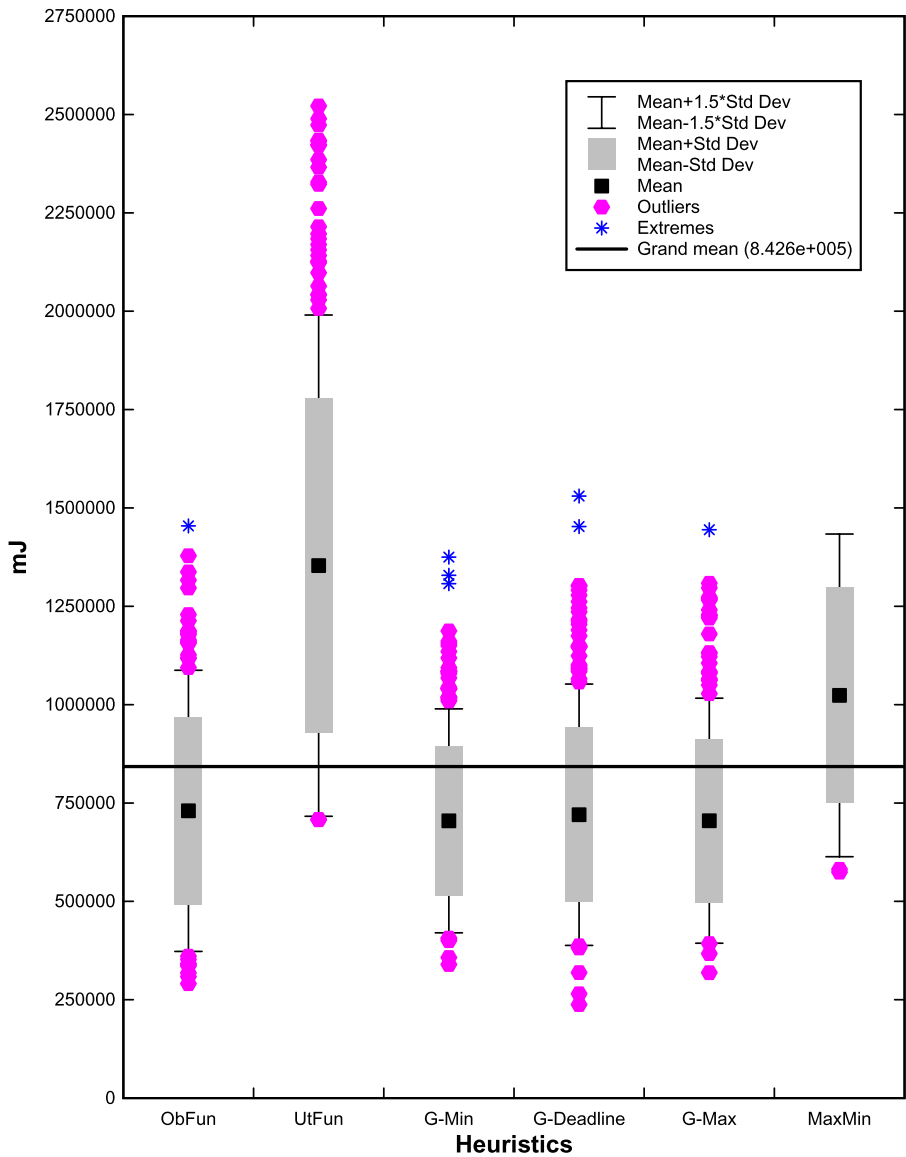


Fig. 13 Energy consumption for 10,000 tasks with $V_{task} = 0.35$ and $V_{PE} = 0.1$

energy consumption and largest mean *makespan*. UtFun continues to demonstrate the same weaknesses observed in all of the other simulations. Because the utility is a function of a PE’s speed and the execution time of the task (19), UtFun does not schedule tasks to the most energy efficient PEs. Based on mean energy consumption and mean *makespan*, G-Deadline had the second best solution. G-Deadline schedules tasks with the most urgent deadline first. If the tasks with the most urgent deadlines

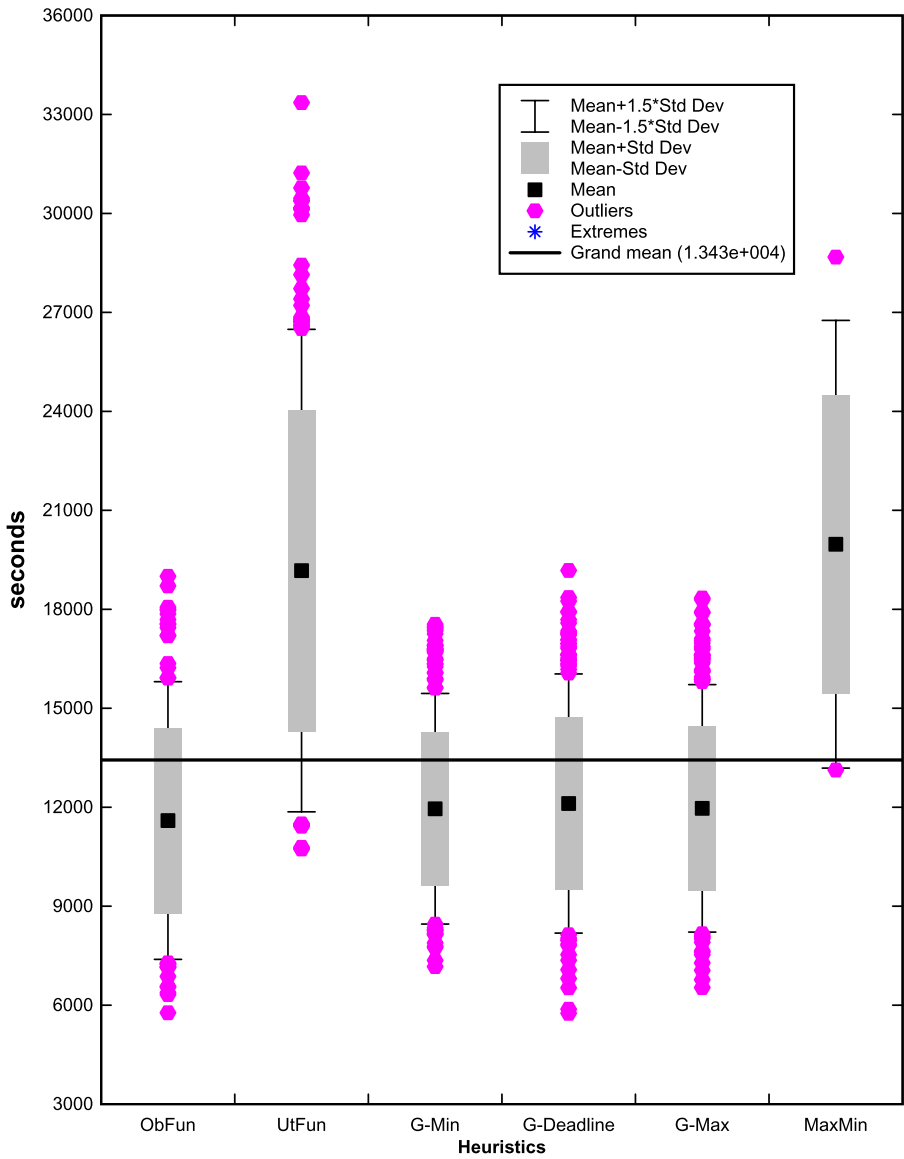


(a) Energy consumption for 10,000 tasks

Fig. 14 10,000 task problem-size

are not scheduled first, then these tasks may need to be scheduled to an inefficient PE to meet its deadline constraint. ObFun, G-Min, G-Deadline, and G-Max all had mean energy consumptions and a mean *makespan* lower than the grand mean.

The run-times of the eight proposed heuristics can be seen in Table 5. Because genetic algorithms rely on evolutionary procedures, such as mutation, crossover, and reproduction, to arrive at an “optimal” solution, their run-times are much higher than

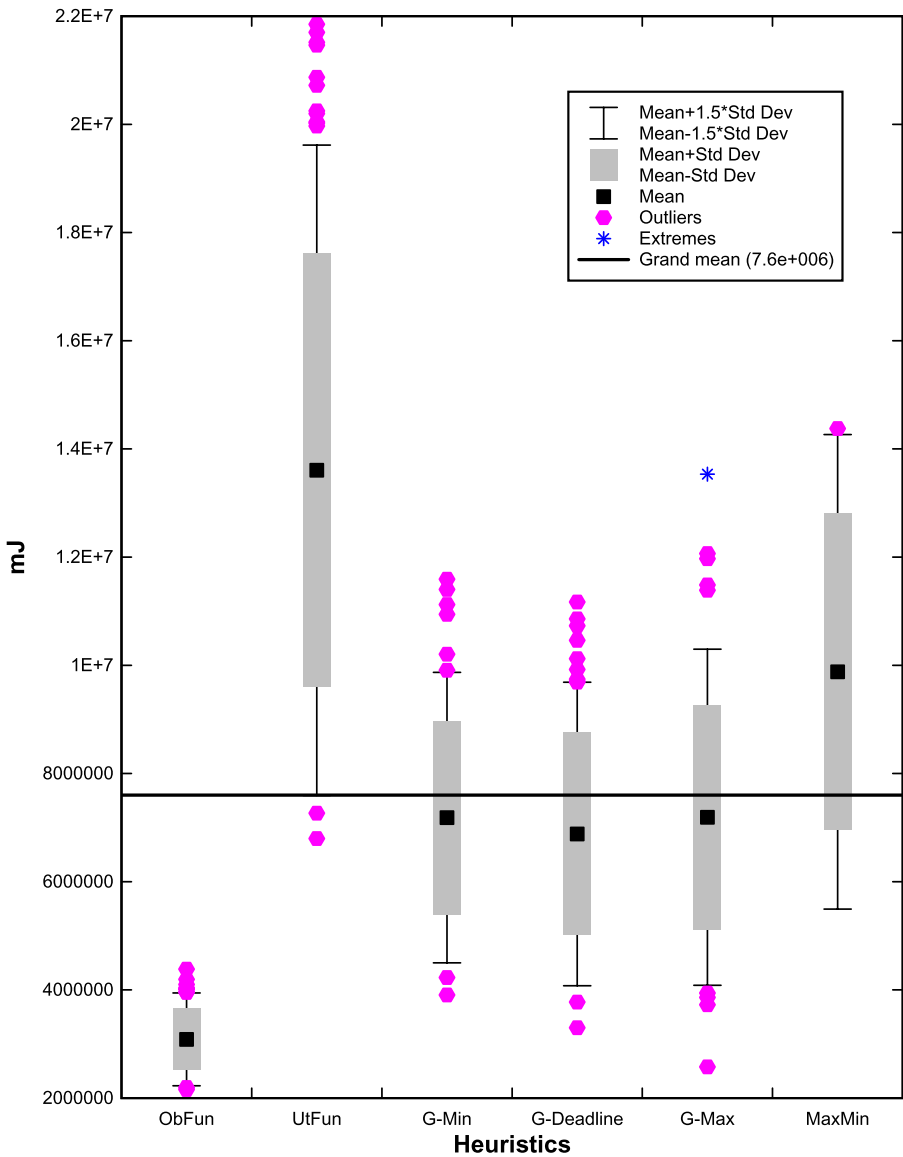


(b) Makespan for 10,000 tasks

Fig. 14 (Continued)

greedy heuristics. G-Min, G-Deadline, and G-Max had the shortest run-times. Note that the GenAlg and GenAlg-DVS heuristics were not executed in the 10,000 and 100,000 task size simulations; therefore, the corresponding entries in Table 5 indicate that these heuristics did not execute (DNE).

Finally, we evaluate the dollar cost. As explained earlier, the total dollar cost constraint, D , was set to \$160 in our simulations. The average dollar cost of \mathcal{PE} for

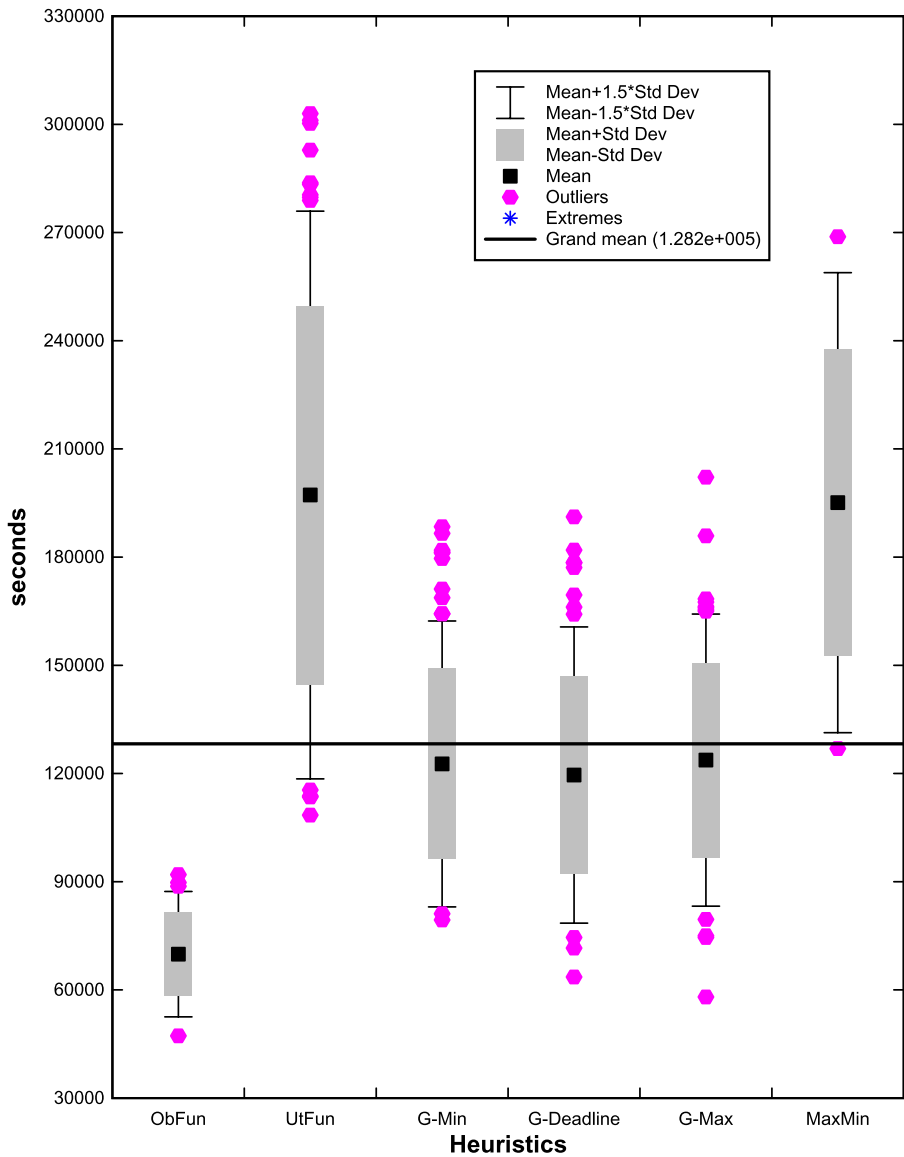


(a) Energy consumption for 100,000 tasks

Fig. 15 100,000 task problem-size

each heuristic is reported in Table 6. All of the heuristics produced very comparable results. UtFun had the lowest average dollar cost by a mere 0.18%.

To summarize, when solving problems with 100 and 1,000 tasks, G-Min, G-Deadline, and G-Max obtained solutions with the lowest energy consumption and the shortest *makespan*. For 10,000 task problems, ObFun, G-Min, G-Deadline, and G-Max demonstrated the highest solution quality. Finally, for the 100,000 task prob-



(b) Makespan for 100,000 tasks

Fig. 15 (Continued)

lems, ObFun vastly out performed all other heuristics. When considering execution times, G-Min, G-Deadline, and G-Max had the best results. All of the proposed heuristics produced results with comparable average dollar costs. Overall, we may conclude ObFun is the best heuristic for large-sized problems, and G-Min, G-Deadline, and G-Max are the best heuristics for small-sized problems.

Table 6 Average dollar cost

| Heuristic | Dollar Cost |
|------------|-------------|
| ObFun | \$156.14 |
| UtFun | \$155.86 |
| G-Min | \$156.66 |
| G-Deadline | \$156.67 |
| G-Max | \$156.28 |
| MaxMin | \$156.64 |
| GenAlg | \$156.87 |
| GenAlg-DVS | \$156.88 |

5 Conclusion

This paper introduced an energy minimizing task scheduling strategy in distributed systems. The problem was formulated as an extension of the Generalized Assignment Problem. Eight heuristics were proposed to solve this problem. Six of these heuristics were greedy heuristics, namely ObFun, UtFun, G-Min, G-Deadline, G-Max, and MaxMin. The last two heuristics were based on naturally inspired genetic algorithms, namely GenAlg and GenAlg-DVS. The eight heuristics were compared against each other with both small and large problem sizes. The simulation results showed that for small-sized problems, G-Min, G-Deadline, and G-Max performed the best. For large-sized problems, ObFun had superior performance in term of mean energy consumption and mean *makespan* against all of the other proposed heuristics.

For the aforementioned problems, we are aware of other possible mathematical optimization techniques that can be utilized, such as dynamic programming, constraint satisfaction, linear programming, trajectory optimization, integer programming, combinatorial optimization, quadratic programming, nonlinear programming, weighted sums of functions, convex programming, normal-boundary intersection, semidefinite programming, homotopy, stochastic programming, game theory, robust programming, infinite-dimensional optimization, constraint programming, calculus of variations, optimal control, goal programming, and multilevel programming. However, we leave that as future work.

References

1. Nathuji R, Isci C, Gokratov E (2007) Exploiting platform heterogeneity for power efficient data centers. In: ICAC '07: proceedings of the fourth international conference on autonomic computing, 2007
2. Heath T, Diniz B, Carrera EV, Wagner M Jr, Bianchini R (2005) Energy conservation in heterogeneous server clusters. In: PPOPP '05: proceedings of the tenth ACM SIGPLAN symposium on principles and practice of parallel programming. ACM Press, New York, pp 186–195
3. Qiu Q, Pedram M (1999) Dynamic power management based on continuous-time Markov decision processes. In: In design automation conference, 1999, pp 555–561
4. Weiser M, Welch B, Demers A, Shenker S (1994) Scheduling for reduced cpu energy. In: OSDI '94: proceedings of the 1st USENIX conference on operating systems design and implementation. USENIX Association, Berkeley, pp 13–23

5. Abdelzaher TF, Lu C (2001) Schedulability analysis and utilization bounds for highly scalable real-time services. In: In IEEE real-time technology and applications symposium, 2001, pp 15–25
6. Bianchini R, Rajamony R (2004) Power and energy management for server systems. *IEEE Comput* 37(11):68–74
7. Elnozahy EM, Kistler M, Rajamony R (2002) Energy-efficient server clusters. In: Proceedings of the 2nd workshop on power-aware computing systems, 2002, pp 179–196
8. Pinheiro E, Bianchini R, Carrera EV, Heath T (2001) Load balancing and unbalancing for power and performance in cluster-based systems. In: Workshop on compilers and operating systems for low power, 2001
9. Hsu C-H, Kremer U (2003) The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. In: Proceedings of ACM SIGPLAN conference on programming language design and implementation. ACM Press, New York, pp 38–48
10. Hwang C-H, Wu AC-H (1997) A predictive system shutdown method for energy saving of event-driven computation. In: 1997 Design automation conference, 1997, pp 28–32
11. Kirovski D, Potkonjak M (1997) System-level synthesis of low-power hard real-time systems. In: DAC '97: proceedings of the 34th annual design automation conference. ACM Press, New York, pp 697–702
12. Dick RP, Jha NK (1997) MOGAC: A multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems. *IEEE Trans Comput-Aided Des Integr Circuits Syst* 17:920–935
13. Schmitz MT, Al-Hashimi BM (2001) Considering power variations of DVS processing elements for energy minimisation in distributed systems. In: Proc. 14th int'l symp. on system synthesis, 2001, pp 250–255
14. Dick RP, Jha NK (2004) COWLS: Hardware-software cosynthesis of wireless low-power distributed embedded client-server systems. *IEEE Trans Comput-Aided Des Integr Circuits Syst* 23(1):2–16
15. Hassani MM, Berangi R (2007) Improving the COWLS algorithm for hardware software co-synthesis of wireless client-server systems using preference vectors and peak power information. In: Proc. int'l conference on computer systems and technologies
16. Dave BP, Lakshminarayana G, Jha NK (1997) COSYN: hardware-software co-synthesis of embedded systems. In: DAC '97: proceedings of the 34th annual design automation conference. ACM Press, New York, pp 703–708
17. Chedid W, Yu C (2005) Power analysis and optimization techniques for energy efficient computer systems. *Adv Comput* 63:129–164
18. Unsal OS, Koren I (2003) System-level power-aware design techniques in real-time systems. *Proc IEEE* 91(7):1055–1069
19. Venkatachalam V, Franz M (2005) Power reduction techniques for microprocessor systems. *ACM Comput Surv* 37(3):195–237
20. Khan SU, Ahmad I (2009) A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids. *IEEE Trans Parallel Distrib Syst* 21(4):346–360
21. Ahmad I, Ranka S, Khan SU (2008) 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, 2008, pp 1–6
22. Luenberger DG (1984) *Linear and nonlinear programming*. Addison-Wesley, Reading
23. Li YA, Antonio JK, Siegel HJ, Tan M, Watson DW (1997) Determining the execution time distribution for a data parallel program in a heterogeneous computing environment. *J Parallel Distrib Comput* 44(1):35–52
24. Ali S, Siegel HJ, Maheswaran M, Ali S, Hensgen D (2000) Task execution time modeling for heterogeneous computing systems. In: HCW '00: proceedings of the 9th heterogeneous computing workshop. IEEE Computer Society, Washington, p 185
25. Papoulis A (1984) *Probability, random variables, and stochastic processes*. McGraw-Hill, New York
26. Yu Y, Prasanna VK (2002) Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In: ICPADS '02: proceedings of the 9th international conference on parallel and distributed systems. IEEE Computer Society, Washington, pp 341–348
27. Khan SU, Ardil C (2009) Energy efficient resource allocation in distributed computing systems. In: International conference on distributed, high-performance and grid computing, 2009, pp 667–673
28. Bodie Z, Marcus A, Kane A (2007) *Essentials of investments*. McGraw-Hill, New York
29. Moler CB (2004) *Numerical computing with Matlab*. Society for Industrial Mathematics

30. Khan SU, Ardil C (2009) On the joint optimization of performance and power consumption in data centers. In: International conference on distributed, high-performance and grid computing, 2009, pp 660–660
31. Siegel HJ, Ali S (2000) Techniques for mapping tasks to machines in heterogeneous computing systems. *J Syst Archit* 46(8):627–639
32. Ali S, Siegel HJ, Maheswaran M, Hensgen D, Ali S (2000) Representing task and machine heterogeneities for heterogeneous computing systems. *Tamkang J Sci Eng* 3(3):195–207
33. Khan SU, Ahmad I (2010) Comparison and analysis of ten static heuristics-based internet data replication techniques. *J Parallel Distrib Comput* 68(2):113–136
34. Khan SU, Ahmad I (2007) A cooperative game theoretical replica placement technique. In: 2007 International conference on parallel and distributed systems
35. Khan SU (2009) 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, 2009