CHAPTER 1

COMPARISON AND ANALYSIS OF GREEDY ENERGY-EFFICIENT SCHEDULING ALGORITHMS FOR COMPUTATIONAL GRIDS

Peder Lindberg[1], James Leingang[1], Daniel Lysaker[1], Kashif Bilal[2], Samee Ullah Khan[1],Pascal Bouvry[3], Nasir Ghani[4], Nasro Min-Allah[2], and Juan Li[1]

[1]North Dakota State University, Fargo, ND 58108-6050, USA
[2]COMSATS Institute of Information Technology, Pakistan
[3]University of Luxembourg, Luxembourg
[4]University of New Mexico, Albuquerque, NM 87131-0001, USA

A computational grid is a distributed computational network, enabled with software that allows cooperation and the sharing of resources. The energy consumption of these large-scale distributed systems is an important problem. As our society becomes more technologically advanced, the size of these computational grids and energy consumption continue to increase. In this paper, we study the problem of optimizing energy consumption and makespan by focusing on different techniques to schedule the tasks to the computational grid. A computational grid is simulated using a wide range of task heterogeneity and size variety. The heuristics are used with the simulated computational grid and the results are compared extensively against each other.

1.1 INTRODUCTION

A computational grid is a distributed computational network enabled with software that allows cooperation and the sharing of resources. The energy

Heuristics for Optimizing Makespan and Energy Consumption in Computational Grids. By Peder Lindberg, James Leingang, Daniel Lysaker, Kashif Bilal, Samee Ullah Khan, Pascal Bouvry, Nasir Ghani, Nasro Min-Allah, and Juan Li

consumption of these large-scale distributed systems is an important problem in today's society. As our society becomes more technologically dependent, the size of these computational grids and amount of energy consumed continues to increase [12].

Energy is the amount of power used over a specific time interval. Power and time are the two major factors analyzed to reduce energy consumption in a computational grid. Power is defined as the rate the distributed system consumes electrical energy during operation. There are a few common ways (listed below) to help minimize the energy consumption of these large-scale distributed systems.One method is Dynamic Voltage Scaling (DVS) [32]. DVS is used to reduce power by scaling down each processing element's (PE) supply voltage (V_{dd}) to one of a few discrete (V_{dd}) levels. Reducing power consumption will increase the execution time of a task on a PE, however, the overall energy consumption will decrease because less power is consumed.

Another method is Dynamic Frequency Scaling (DFS). DFS can be used either for energy conservation or lowering the heat produced by a processor by lowering the frequency at which the processor can issue instructions. Lowering the frequency will increase the amount of time a PE needs to complete a task, but this leads to energy conservation. Energy is conserved with DFS because the PE consumes less power when running at lower frequencies.

A third technique commonly used to conserve energy on the PE level is clock gating. Clock gating adds additional logic to the PE that disables certain portions of the switching activity from changing states. This method reduces the power by preventing the PE from constantly switching, that accounts for a large fraction of the PE's energy consumption.

We chose to use the DVS approach because the approach: (a) accurately simulates real world problems, (b) has no switching, and (c) has a closed form relationship between voltage, power, energy and makespan.

In this paper, we study and analyze seven greedy heuristics-based algorithms. The greedy heuristics are used to find solutions for the energy-aware task allocation (EATA) problem of assigning a group of tasks to a set of PEs. Each greedy heuristic is given the same conditions and parameters to maintain a fair comparison. The proposed heuristics are: Greedy-Min, Greedy-Max, Greedy-Deadline, MaxMin, MinMin StdDev, MinMax StdDev, and ObFun.

Each PE in a large-scale computational grid is composed of many hardware devices (hard drives, memory, communication links, etc.) that contribute to the total energy consumption. The presented heuristics provide enough accuracy to give a good estimate of the total energy consumption while still allowing one to simulate a large-scale date center [15].

Initially, Greedy-Min, Greedy-Max, Greedy-Deadline, MaxMin, MinMin StdDev, and MinMax StdDev *rearrange* the tasks so that they are in the order they will be distributed to PEs. Greedy-Min schedules the tasks with the shortest completion times first. Greedy-Max schedules the tasks with the longest completion times first. Greedy-Deadline schedules the tasks with the most urgent deadlines first. MaxMin initially schedules the tasks to the

least efficient PEs to allow easier scheduling of the subsequent tasks. MinMin StdDev first schedules the tasks in ascending order and then *rearranges* the tasks in ascending order based on their standard deviation. MinMax StdDev is similar except that the tasks are *rearranged* in descending order after the standard deviation of each task is determined. After a task is assigned to a PE, the PE is set to the minimum DVS level that keeps the task from overshooting the deadline constraint.

ObFun is a greedy heuristic that uses two objective functions to assign tasks to appropriate PEs. The first objective function decides which task will be assigned to a PE during each iteration by examining the run-time and power consumption of each task on every PE. The second objective function determines the most appropriate PE for the task.

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. The number of tasks are also varied from 1000 and 100,000. A detailed explanation of our simulation test-bed will be given in Section 5.

The remainder of this paper is organized as follows. The problem formulation is introduced in Section 2. Next, we discuss the task scheduling heuristics in Section 3. The simulation results are reviewed in Section 4. In Section 5, we present related research. Finally, we present a conclusion in Section 6.

1.2 PROBLEM FORMULATION

1.2.1 The System Model

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

PEs. Let the set of PEs be denoted as, $\mathcal{PE} = \{PE_1, PE_2, ..., PE_m\}$. Each PE is assumed to be equipped with a DVS module which we will describe in the subsequent sections. 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}.

DVS. DVS is a method that can be used to conserve energy in a data center [32]. With the DVS technique, each processing element's (PE) supply voltage (V_{dd}) can be scaled to a discrete number of V_{dd} levels. By decreasing the operational frequency (f) and V_{dd}, the amount of energy conserved may be increased. A PE will complete fewer computational cycles while operating at a lower frequency; therefore, decreasing the frequency increases the makespan. The makespan is defined as the amount of time to complete all the tasks given

to the data center. The following equations give the relationship between f, power consumption, and energy consumption over the period $[0, T]$:

$$f = \frac{k \cdot (V_{dd} - V_t)^2}{V_{dd}}, \tag{1.1}$$

$$P = C_L \cdot N_{0\to1} \cdot f \cdot V_{dd}^2, \tag{1.2}$$

$$E = \int_0^T P(t)\, dt, \tag{1.3}$$

where C_L is the switching capacitance, $N_{0\to1}$ 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.

Tasks. A metatask, $T = \{t_1, t_2, ..., t_n\}$, 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.

1.2.2 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,

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

subject to the following constraints:

$$x_{ij} \in 0, 1, i = 1, 2, ..., n; j = 1, 2, ..., m, \tag{1.4}$$

$$t_i \rightarrow m_j, \forall i, \forall j; \text{ if } m_{PE_j} > m_{t_i}; \text{ then } x_{ij} = 1, \tag{1.5}$$

$$t_{ij} x_{ij} \leq d_i, \forall i, \forall j, x_{ij} = 1, \tag{1.6}$$

$$(t_{ij} x_{ij} \leq d_i) \in 0, 1, \tag{1.7}$$

$$\prod_{i=1}^{n} (t_{ij} x_{ij} \leq d_i) = 1, \forall i, \forall j, x_{ij} = 1, \tag{1.8}$$

Constraint 1.4 is the mapping constraint. t_i is assigned to PE_j when $x_{ij} = 1$. Constraint 1.5 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 1.6 relates to the fulfillment of the deadline of each task. Constraint 1.7 shows there is a Boolean relationship between the deadline and the actual execution time of the tasks. Constraint 1.8 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.

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 1.6, 1.7, and 1.8. 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 [23].

1.3 PROPOSED ALGORITHMS

In this section we will describe the inner workings of our seven proposed heuristics.

All of the task execution times are obtained from an estimated time of completion (ETC) matrix [22]. 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 [3]. There are three major parameters that determine the heterogeneity of the ETC matrix:

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}.

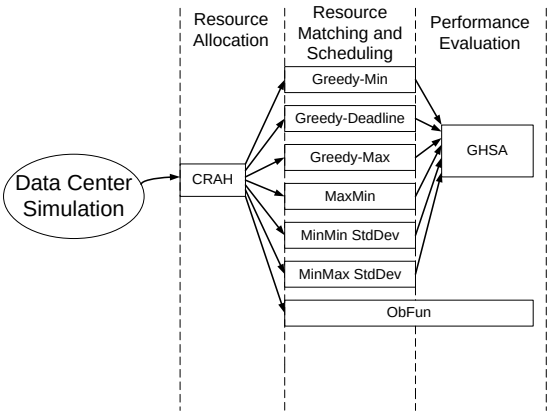


Figure 1.1 Simulation Flow Chart

Because CVB uses a gamma distribution [26], 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

$$\alpha_{task} = 1/V_{task}^2, \tag{1.9}$$

$$\alpha_{PE} = 1/V_{PE}^2, \tag{1.10}$$

$$\beta_{task} = \mu_{task}/\alpha_{task}, \tag{1.11}$$

$$\beta_{PE} = G(\alpha_{task}, \beta_{task})/\alpha_{PE}, \tag{1.12}$$

where $G(\alpha_{task}, \beta_{task})$ 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

$$d_i = \frac{|t_i|}{|\mathcal{PE}|} \cdot arg_j max(t_{ij}) \cdot k_d, \tag{1.13}$$

where k_d is a parameter that can tighten d_i [33], [20].

1.3.1 Greedy Heuristics

The resource allocation for the following six greedy heuristics is achieved by CRAH. Algorithm 1 shows the pseudo-code for CRAH. The CRAH algorithm

Input: ETC, $\mathcal{PE}_p$, d_i $\forall t_i \in T$
Output: T to $\mathcal{PE}$ mapping, $\mathcal{E}_{min}$, M
INVOKE Greedy Heuristic to *rearrange* ETC and generate EEC;
while $k < k_{max}$ **do**
    Generate Random $\mathcal{PE}$;
    CALCULATE E_{sol};
    $E_{min} \leftarrow E_{sol}$;
    repeat
        $E'_{min} \leftarrow E_{min}$;
        foreach $PE_j \in \mathcal{PE}_p$ **do**
            Add PE_j to $\mathcal{PE}$;
            CALCULATE E_{sol};
            if $E_{sol} < E_{min}$ **then** $E_{min} \leftarrow E_{sol}$ Remove PE_j from $\mathcal{PE}$;
        end
        foreach $PE_j \in \mathcal{PE}$ **do**
            Remove PE_j from $\mathcal{PE}$;
            CALCULATE E_{sol};
            if $E_{sol} < E_{min}$ **then** $E_{min} \leftarrow E_{sol}$ Add PE_j to $\mathcal{PE}$;
        end
        if $E_{min} \geq \mathcal{E}_{min}$ **then**
            INCREMENT k
        else
            $k \leftarrow 0$;
            $\mathcal{E}_{min} \leftarrow E_{min}$
        end
    until $E_{min} \geq E'_{min}$;
end

Algorithm 1: Constructive resource allocation heuristic (CRAH)

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 1.2 illustrates one method of *rearranging* the ETC matrix. Figure 1.2(a) shows the original ETC matrix. The rows are sorted in ascending order (Figure 1.2(b)). Next, the rows of the ETC matrix are swapped such that the execution times in the first column are arranged in ascending order (Figure 1.2(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 Figures 1.2(d)-(f). Next, an estimated energy consumption (EEC) matrix is generated by multiplying the PE's instantaneous power consumption by the task's 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 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 Line 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.

1.3.1.1 Greedy Heuristic Scheduling Algorithm The greedy heuristic scheduling algorithm (GHSA) performs the task scheduling for Greedy-Min, Greedy-Deadline, Greedy-Max, MaxMin, MinMin Std Dev, and MinMax StdDev. The major difference among the six greedy heuristics is how these heuristics schedule T to $\mathcal{PE}$. Algorithm 1 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 1.1). The DVS_k is incrementally increased until d_i is met. 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

Input: ETC, $\mathcal{PE}_p$, d_i $\forall t_i \in T$, and $\mathcal{PE}$
Output: T to $\mathcal{PE}$ mapping, E_{sol}, M
foreach $t_i \in T$ **do**
  foreach $PE_j \in \mathcal{PE}$ **do**
    for $DVS_k = 1$ *to* 4 **do**
      if $t_{ijk} + m_j \leq d_i$ **then**
        Assign t_i to PE_j at DVS_k;
        $m_j \leftarrow m_j + ETC(ij)$;
        $E_{sol} \leftarrow E_{sol} + EEC(ij)$;
      end
    end
    if t_i not assigned **then**
      $d_{flag} \leftarrow 1$;
      EXIT;
    end
  end
end
foreach $PE_j \in \mathcal{PE}$ **do**
  $E_{sol} \leftarrow E_{sol} + E_{idle}$;
end

Algorithm 2: Greedy heuristic scheduling algorithm (GHSA)

Table 1.1 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

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,

$$E_{idle} = p_j \cdot t_{idle} \cdot k_{idle}, \tag{1.14}$$

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.

1.3.1.2 Greedy-Min The Greedy-Min heuristic (Algorithm 3) schedules the tasks with the shortest execution times first. The motivation behind schedul-

$$\begin{vmatrix} 8 & 7 & 10 \\ 10 & 9 & 5 \\ 6 & 12 & 7 \end{vmatrix} \quad \begin{vmatrix} 7 & 8 & 10 \\ 5 & 9 & 10 \\ 6 & 7 & 12 \end{vmatrix} \quad \begin{vmatrix} 5 & 9 & 10 \\ 6 & 7 & 12 \\ 7 & 8 & 10 \end{vmatrix}$$

(a) (b) (c)

ETC Matrices

$$\begin{vmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{vmatrix} \quad \begin{vmatrix} 2 & 1 & 3 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{vmatrix} \quad \begin{vmatrix} 3 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{vmatrix}$$

(d) (e) (f)

Index Matrices

Figure 1.2 ETC matrix *rearranged* for Greedy-Min

Input: *ETC*
Output: *ETC*, *EEC*, *I-ETC*
foreach $R \in ETC$ **do**
  Sort R in ascending order;
  Sort corresponding row in *I-ETC*;
end
$\forall R \in ETC$, swap R such that C_1 is in ascending order;
Apply same changes to *I-ETC*;

Algorithm 3: Greedy-Min

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

Algorithm 4: Greedy-Deadline

ing 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. Greedy-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 that Greedy-Min, Greedy-Deadline, Greedy-Max, and MaxMin all have the same inputs and outputs. Figure 1.2 illustrates the process of *rearranging* the ETC matrix. Greedy-Min *rearranges* the ETC matrix.

Deadlines	
t_1	13
t_2	15
t_3	10

(a)

Deadlines

$$\left|\begin{matrix} 8 & 7 & 10 \\ 10 & 9 & 5 \\ 6 & 12 & 7 \end{matrix}\right| \quad \left|\begin{matrix} 7 & 8 & 10 \\ 5 & 9 & 10 \\ 6 & 7 & 12 \end{matrix}\right| \quad \left|\begin{matrix} 6 & 7 & 12 \\ 7 & 8 & 10 \\ 5 & 9 & 10 \end{matrix}\right|$$

(b) (c) (d)

ETC Matrix

Figure 1.3 ETC matrix *rearranged* for Greedy-Deadline

$$\left|\begin{matrix} 8 & 7 & 10 \\ 10 & 9 & 5 \\ 6 & 12 & 7 \end{matrix}\right| \quad \left|\begin{matrix} 7 & 8 & 10 \\ 5 & 9 & 10 \\ 6 & 7 & 12 \end{matrix}\right| \quad \left|\begin{matrix} 7 & 8 & 10 \\ 6 & 7 & 12 \\ 5 & 9 & 10 \end{matrix}\right|$$

(a) (b) (c)

ETC Matrix

Figure 1.4 ETC matrix *rearranged* for Greedy-Max

1.3.1.3 Greedy-Deadline One of the major differences between Greedy-Deadline and Greedy-Min is in the task scheduling. In Greedy-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 1.3 shows how the ETC matrix is *rearranged*. As seen in Figure 1.3(c), the rows are sorted in ascending order. In Figure 1.3(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 Greedy-Deadline *rearranges* the ETC matrix, GHSA is invoked.

1.3.1.4 Greedy-Max In Greedy-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 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 1.4 demonstrates the process of *rearranging* the ETC matrix for Greedy-Max. In Figure 1.4(b), the rows of the ETC matrix are sorted in ascending order. In Figure 1.4(c), the rows are swapped so that the execution times in the first column are arranged in descending order.

Input: ETC
Output: ETC, EEC, $I\text{-}ETC$
foreach $R \in ETC$ **do**
  Sort R in ascending order;
  Sort corresponding row in $I\text{-}ETC$;
end
$\forall R \in ETC$, swap R such that C_1 is in descending order;
Apply same changes to $I\text{-}ETC$;
INVOKE GHSA;

Algorithm 5: Greedy-Max

$$\begin{vmatrix} 8 & 7 & 11 \\ 10 & 9 & 5 \\ 6 & 12 & 7 \end{vmatrix} \quad \begin{vmatrix} 11 & 8 & 7 \\ 10 & 9 & 5 \\ 12 & 7 & 6 \end{vmatrix} \quad \begin{vmatrix} 10 & 9 & 5 \\ 11 & 8 & 7 \\ 12 & 7 & 6 \end{vmatrix}$$

(a) (b) (c)

ETC Matrix

Figure 1.5 ETC matrix *rearranged* for MaxMin

Input: ETC
Output: ETC, EEC, $I\text{-}ETC$
foreach $R \in ETC$ **do**
  Sort R in descending order;
  Sort corresponding row in $I\text{-}ETC$;
end
$\forall R \in ETC$, swap R such that C_1 is in ascending order;
Apply same changes to $I\text{-}ETC$;
INVOKE GHSA;

Algorithm 6: MaxMin

1.3.1.5 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 1.5 shows the process of *rearranging* the ETC matrix. In Figure 1.5(b), the rows of the ETC matrix are sorted in descending order. In Figure 1.5(c), the rows are swapped so that the execution times in the first column are arranged in descending order.

1.3.1.6 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.

Input: ETC, $\mathcal{PE}_p$, d_i $\forall t_i \in T$, and $\mathcal{PE}$
Output: T to $\mathcal{PE}$ mapping, E_{sol}, M
foreach $t_i \in T$ **do**
  Calculate TS_i;
end
Sort TS in descending order;
foreach $t_i \in TS$ **do**
  foreach $PE_j \in \mathcal{PE}$ **do**
    Calculate PS_{ij};
  end
  $j \leftarrow arg_j min(PS_{ij})$;
  for $DVS_k = 1$ *to* 4 **do**
    if $t_{ijk} + m_j \leq d_i$ **then**
      Assign t_i to PE_j at DVS_k;
      $m_j \leftarrow m_j + ETC(ij)$;
      $E_{sol} \leftarrow E_{sol} + EEC(ij)$;
    end
  end
  if t_i not assigned **then**
    $d_{flag} \leftarrow 1$;
    EXIT;
  end
end
foreach $PE_j \in \mathcal{PE}$ **do**
  $E_j \leftarrow E_j + E_{idle}$;
end

Algorithm 7: ObFun

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

$$TS_i = \alpha_1(T_{2,i} - T_{1,i}) + \alpha_2(P_{2,j} - P_{1,j}) + \alpha_3 \frac{T_{1,i} + T_{2,i}}{\dfrac{tasks}{\sum_{k=1} (T_{1,k} + T_{2,k})}} + \alpha_4 + \alpha_5 + \alpha_6, \tag{1.15}$$

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.

Table 1.2 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

- 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 1.2.

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:

$$PS = \beta_1 T_{1,PE_j,t_i} + \beta_2 P_{1,PE_j,t_i} + \beta_3 load(PE_j), \tag{1.16}$$

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 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.

1.3.1.7 MinMin StdDev The MinMin StdDev heuristic (Algorithm 8) schedules tasks with the shortest execution times first, then the rows are *rearranged* in ascending order based on each row's standard deviation. The motivation

Input: ETC
Output: ETC, EEC, $I\text{-}ETC$
foreach $R \in ETC$ **do**
    Find standard deviation of each $t_i \in R_i$;
    Sort R in ascending order according to each t_i's standard deviation;
    Sort corresponding row in $I\text{-}ETC$;
end
$\forall R \in ETC$, swap R such that C_1 is in ascending order;
Apply same changes to $I\text{-}ETC$;
INVOKE GHSA;

Algorithm 8: MinMin StdDev

$$\left|\begin{matrix} 5 & 13 & 7 \\ 10 & 9 & 8 \\ 6 & 12 & 7 \end{matrix}\right| \quad \left|\begin{matrix} 5 & 7 & 13 \\ 8 & 9 & 10 \\ 6 & 7 & 12 \end{matrix}\right| \quad \left|\begin{matrix} 8 & 9 & 10 \\ 6 & 7 & 12 \\ 5 & 7 & 13 \end{matrix}\right|$$

(a) (b) (c)

ETC Matrix

Figure 1.6 ETC matrix *rearranged* for MinMin StdDev

Input: ETC
Output: ETC, EEC, $I\text{-}ETC$
foreach $R \in ETC$ **do**
    Find standard deviation of each $t_i \in R_i$ Sort R in descending order according to each t_i's standard deviation;
    Sort corresponding row in $I\text{-}ETC$;
end
$\forall R \in ETC$, swap R such that C_1 is in ascending order;
Apply same changes to $I\text{-}ETC$;
INVOKE GHSA;

Algorithm 9: MinMax StdDev

behind this algorithm is to schedule the tasks with the most consistent execution run first. Such a method will provide consistent results each time the algorithm is run and also will induce slack at the end of the algorithm to schedule the tasks with more inconsistent run-times. Figure 1.6 shows how the ETC matrix is *rearranged*. In Figure 1.6(b), the rows are sorted in ascending order. The standard deviation function is then run to find the standard deviation of each row, then the rows are *rearranged* based on these values as seen in Figure 1.6(c). The first row of Figure 1.6(c) has a standard deviation of .67, the 2nd row has a standard deviation of 1.82 and the third row's standard deviation is 2.78.

$$\begin{vmatrix} 5 & 13 & 7 \\ 10 & 9 & 8 \\ 6 & 12 & 7 \end{vmatrix} \quad \begin{vmatrix} 5 & 7 & 13 \\ 8 & 9 & 10 \\ 6 & 7 & 12 \end{vmatrix} \quad \begin{vmatrix} 5 & 7 & 13 \\ 6 & 7 & 12 \\ 8 & 9 & 10 \end{vmatrix}$$

(a) (b) (c)

ETC Matrix

Figure 1.7 ETC matrix *rearranged* for MinMax StdDev

Table 1.3 Summary of system parameters

System Parameters	
μ_{task}	10
V_{task}	{0.1, 0.15, 0.35}
V_{PE}	{0.1, 0.15, 0.35}
k_d	{1, 1.3, 1.8}
$\lvert\mathcal{PE}\rvert$	16
$\lvert T\rvert$	{1,000, 10,000, 100,000}
DVS Levels	4

1.3.1.8 MinMax StdDev MinMax StdDev (Algorithm 9) runs very similarly to MinMin StdDev. One major difference is that the rows are *rearranged* in descending order based on the standard deviation of each row. There are two advantages that come from *rearranging* the rows in this manner. Because the rows are arranged in ascending order and the row with the highest standard deviation is scheduled first, the first task will be scheduled on a PE where it has the shortest projected run-time. It follows that the subsequent tasks will be more consistent in run-time over the span of PE's due to the low standard deviation of the row. Such an algorithm is able to perform well in the case where many of the PEs are at full capacity and the algorithm must search to find an available PE. Figure 1.7(c) shows how the rows are *rearranged* in descending order based on their standard deviation.

1.4 SIMULATIONS, RESULTS, AND DISCUSSION

All of the heuristics introduced in this paper were implemented in Matlab. Matlab can efficiently perform operations on large matrices [24]. 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 seven introduced scheduling heuristics.
2. To measure the impact of system parameter variation.

1.4.1 Workload

For the workload, we obtained task characteristics from an ETC matrix. An explanation of the generation of our CVB ETC matrix was detailed in Section 3. 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 and 0.35. These values were chosen to incorporate variance in our task execution times and are supported in previous studies [3], [19], and [2]. The deadline, d_i, of each t_i is based on the ETC matrix and given by Equation 1.13. To vary the heterogeneity of d_i, the k_d parameter in Equation 1.13 is varied from 1 to 1.8. For small-size problems, the number of tasks was varied from 1000 to 1,000 and the number of PEs was set to 16 [29]. 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 [4]. Therefore, if one must have 256 PEs to choose from, then they must have at least 500,000 tasks to solve. 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 [15], [17], [18], and [16]. 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 1.3.

1.4.2 Comparative results

1.4.2.1 Small-size Problems The simulation results for the small-size problems are shown in Figures 1.8(a)-1.8(b). These figures show the average energy consumption and *makespan* of the seven proposed heuristics. To thoroughly benchmark our heuristics, we varied the simulation system parameters considerably in order to compile a wide range of data. The V_{task}, V_{PE}, and k_d parameters each have three possible values as observed in Table 1.3. 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 1.3. To gain confidence in our results, the simulations 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 ±1.5 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. Outliers and Extremes mark results that fall outside of ±1.5 times the standard deviation. In the subsequent text we will discuss the results for 1000, 1,000, 10,000, and 100,000 tasks.

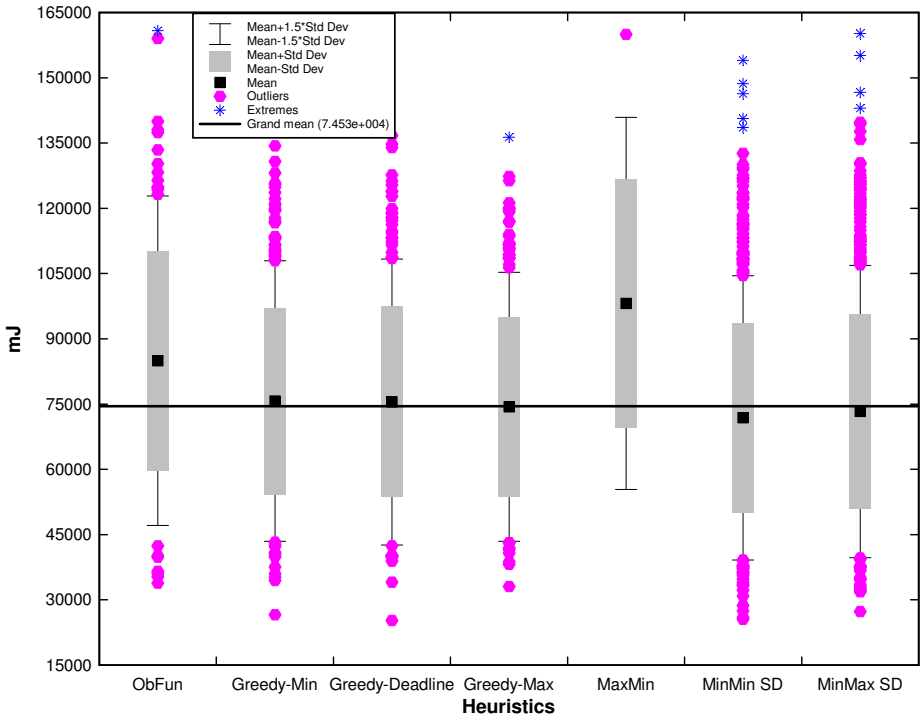
1,000 tasks:

Figure 1.8(a) shows the energy consumption for 1,000 task problems. We can notice that Greedy-Max no longer has the lowest mean energy. MinMin StdDev now has the lowest mean energy consumption (3.39% lower than Greedy-Max.) Observe that the two Standard Deviation Heuristics are the only heuristics that clearly better than the grand mean. Greedy-Max is the only other Heuristic that consumes less energy than the Grand Mean. MinMin StdDev and MinMax StdDev also display the widest range of results observed. This range can be seen in Figure 1.8(a).

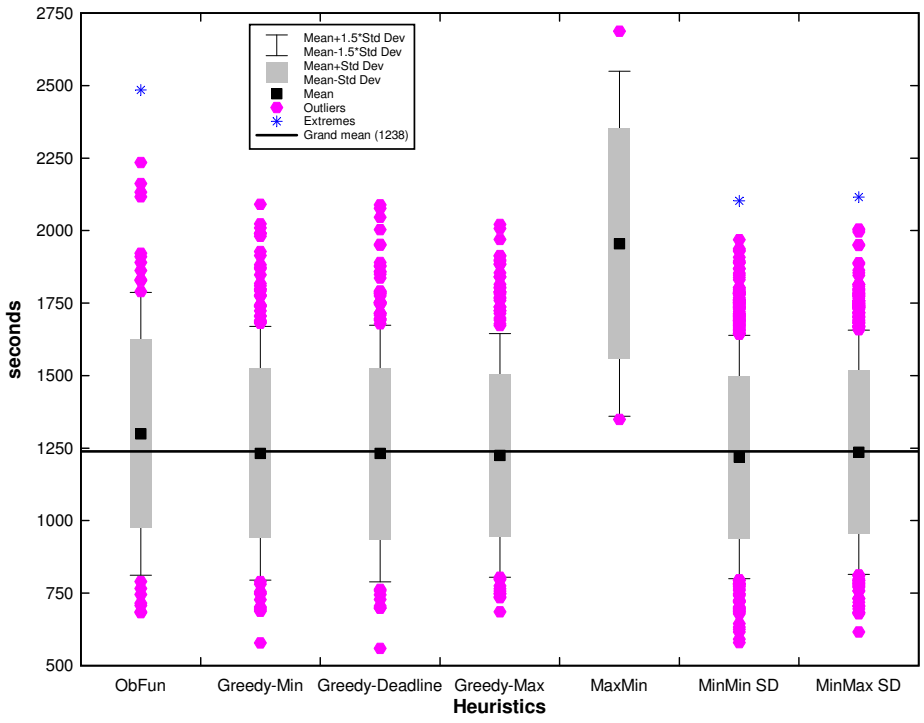
Figure 1.8(b) depicts the *makespan* of the seven heuristics. Greedy-Min, Greedy-Deadline, Greedy-Max, MinMin StdDev and MinMax StdDev all had a mean *makespan* within 0.89% of each other, so there is not one heuristics that is significantly better than the others. MaxMin again performed poorly compared to the rest. ObFun continues to improve as the number of tasks increase, but still has a higher *makespan* than the grand mean. 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 ($V_{task} = V_{PE} = 0.35$). MinMin StdDev performed 21% better than Greedy-Max and 7.9% better than MinMax StdDev which was the heuristic that had the closest *makespan* in this case. These results are depicted in Figure 1.9(a). When there is a high degree of heterogeneity in the ETC matrix, there are more tasks with longer execution times. In such a circumstance, MinMin StdDev is designed to perform consistently better than the rest. Observe that almost the entire range of MinMin StdDev falls below the grand mean for a case with a high degree of heterogeneity.

1.4.2.2 Large-size Problems **10,000 task problem size:**

Figure 1.10(a) shows that there are six heuristics with highly comparable results, namely Greedy-Min, Greedy-Deadline, Greedy-Max, ObFun, MinMin StdDev and MinMax StdDev. MinMin StdDev obtained a mean energy consumption only 3.96% greater than the Greedy-Min. We also can observe that as the problem size increases, ObFun performs better. In certain cases ObFun obtained the lowest mean energy consumption compared to the other heuristics by up to 20% of the mean. Figure 1.9(b) illustrates the mean energy

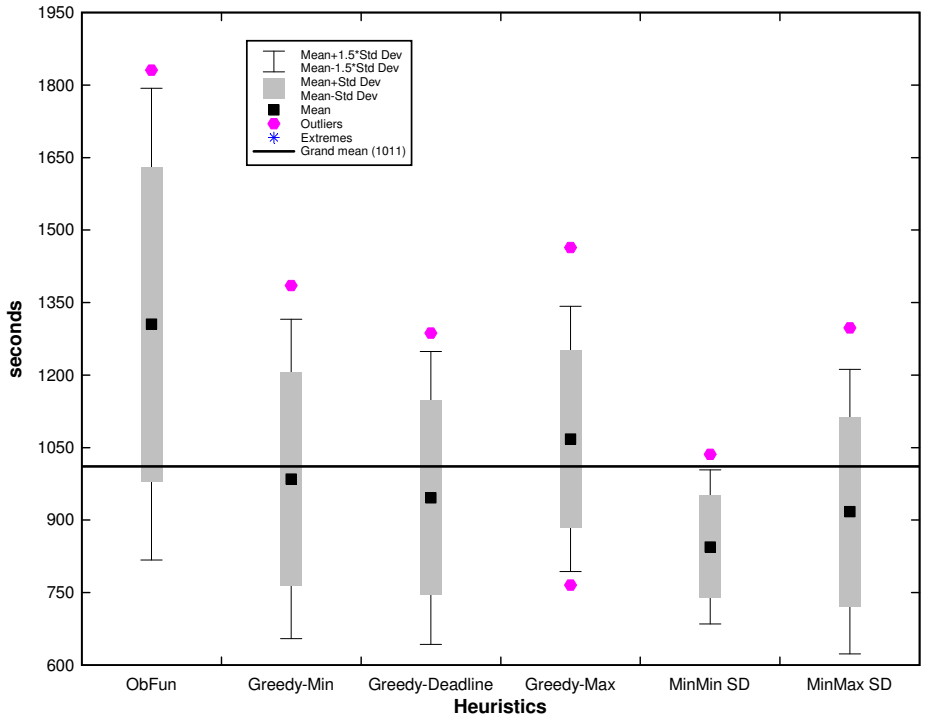


(a) Energy consumption for 1,000 tasks

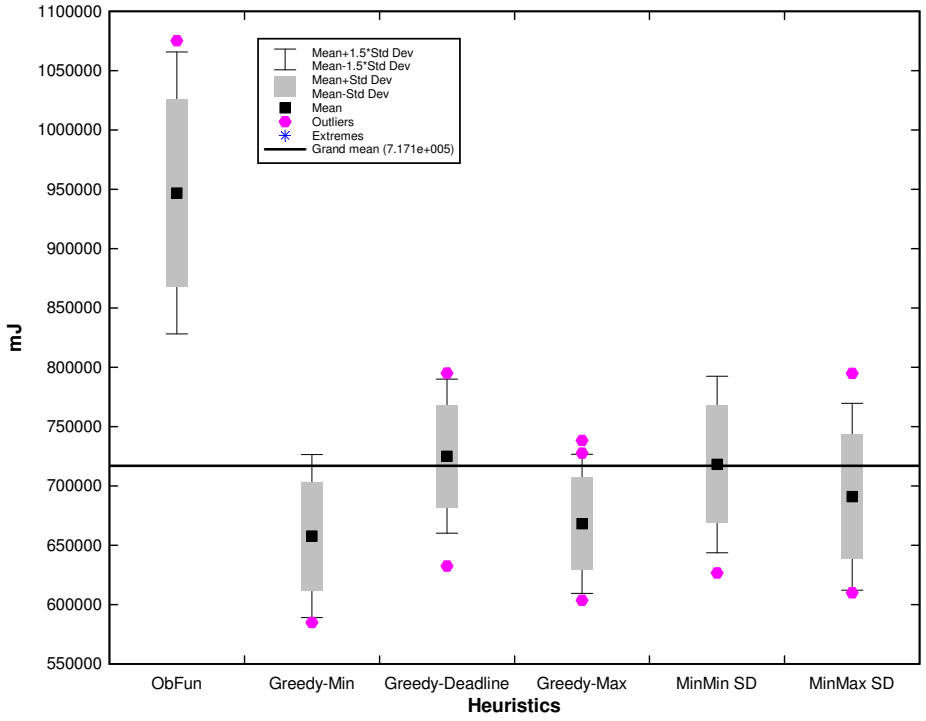


(b) Makespan for 1,000 tasks

Figure 1.8 1,000 task problem-size

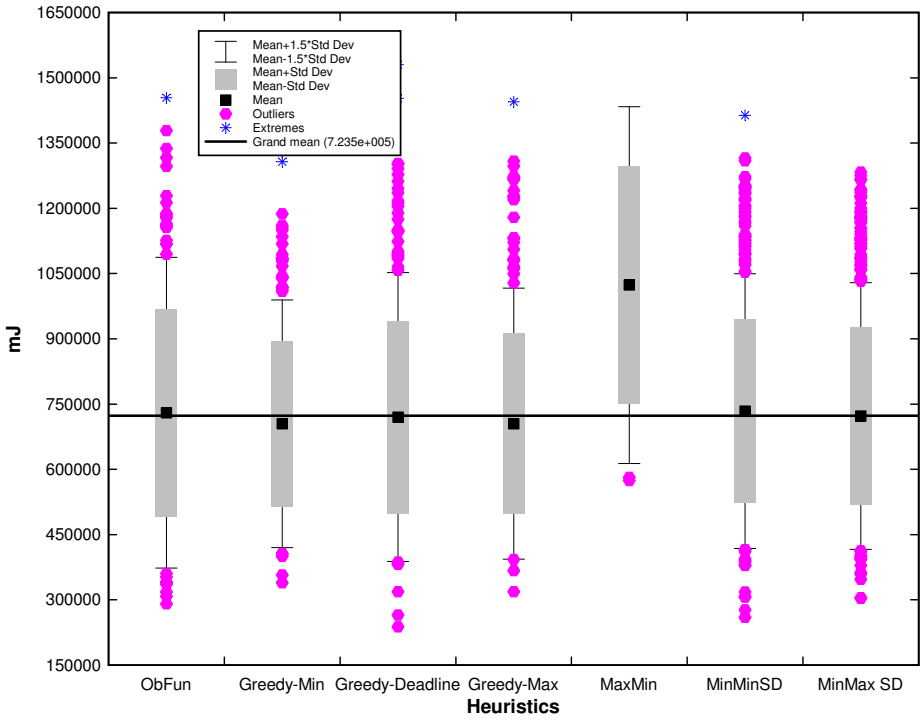


(a) Energy consumption for 1,000 tasks with $k_d = 1.8$ and high heterogeneity ($V_{task} = V_{PE} = 0.35$)

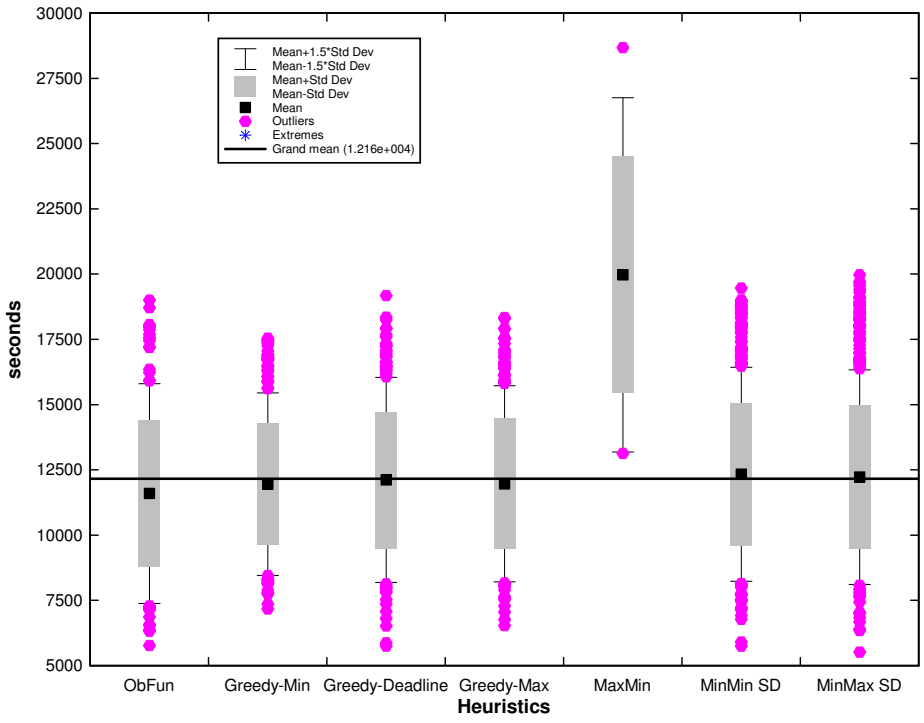


(b) Energy consumption for 10,000 tasks with $V_{task} = 0.35$ and $V_{PE} = 0.1$

Figure 1.9 1,000 and 10,000 case examples



(a) Energy consumption for 10,000 tasks



(b) Makespan for 10,000 tasks

Figure 1.10 10,000 task problem-size

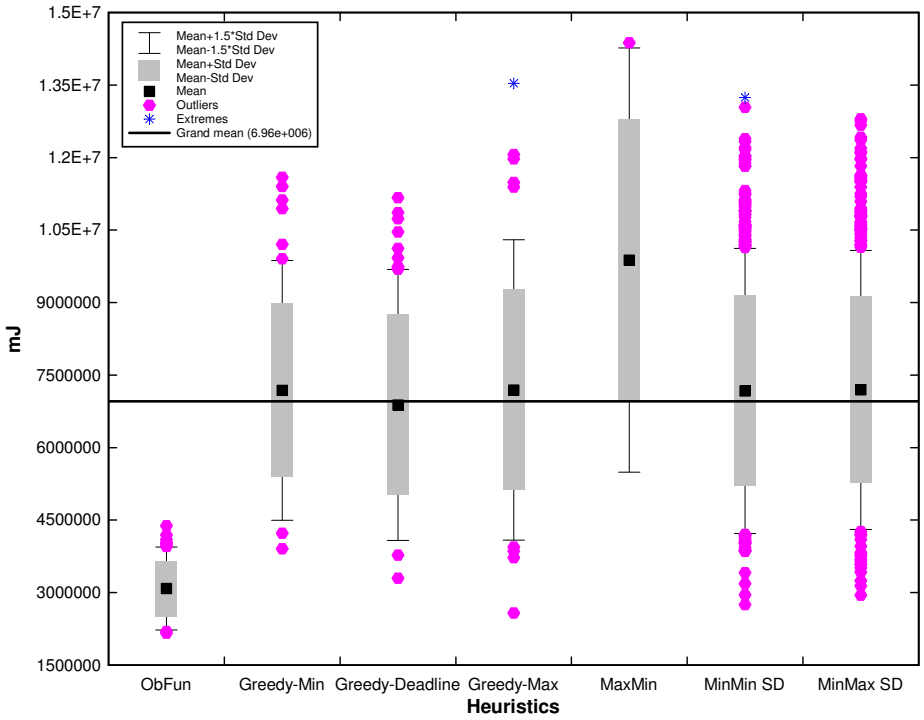
consumption when V_{PE} is set to 0.1, V_{task} is set to 0.1 and k_d was set to 1.3. In the above case, ObFun had a mean energy consumption 23.4% higher than any other Heuristic. When there is low task heterogeneity, the objective function used in ObFun (Equation 1.15), is especially ineffective. In the case of high heterogeneity, ObFun usually outperforms the other six heuristics. Equation 1.15 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 it's objective function, ObFun produces better results in the case of high heterogeneity and worse results in the case of low heterogeneity.

The plot in Figure 1.10(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 induces 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. Our results show that heuristics that initially assign tasks to the most efficient PEs exhibit better results.

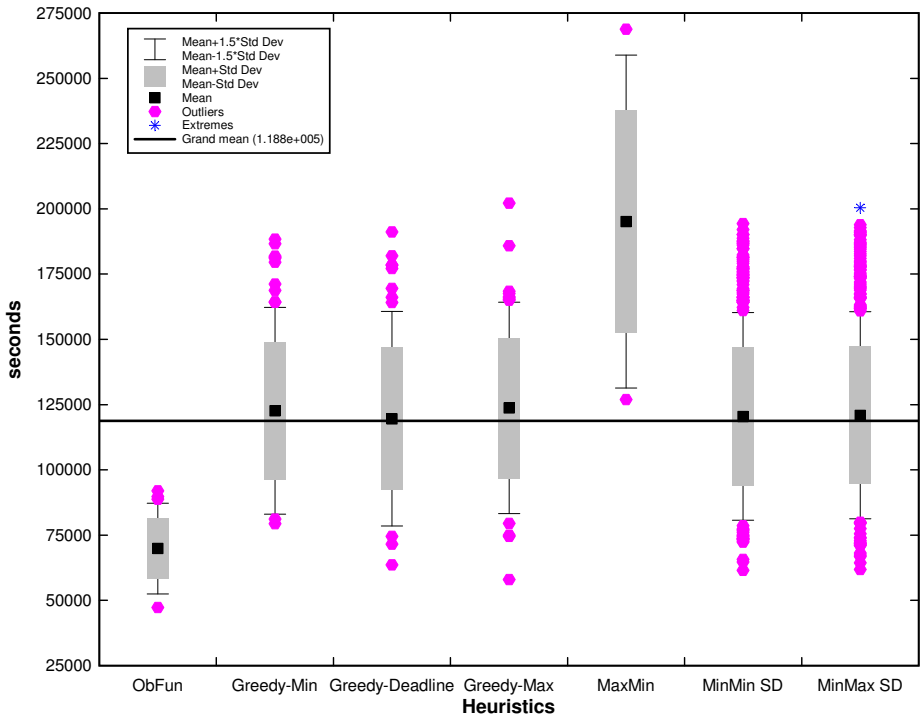
100,000 tasks:

Figures 1.11(a) and 1.11(b) details the energy consumption and *makespan* of the seven 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 MaxMin had the largest mean energy consumption and largest mean *makespan*. MaxMin continues to demonstrate the same weaknesses observed in all of the other simulations. Based on mean energy consumption and mean *makespan*, Greedy-Deadline had the second best solution. Greedy-Deadline schedules tasks with the most urgent deadline first. If the tasks with the most urgent deadlines are not scheduled first, then these tasks may need to be scheduled to an inefficient PE to meet it's deadline constraint. Because of the dominance of ObFun, Greedy-Min, Greedy-Max, MinMin StdDev, MinMax StdDev and MaxMin all had mean energy consumptions and a mean *makespan* higher than the grand mean. In rare cases, the outliers of Greedy-Max, MinMin StdDev, and MinMax StdDev can compete with the mean of ObFun with Greedy-Max coming the closest. Even in the best case scenario for Greedy-Max (low heterogeneity) doesn't compete with ObFun.

The run-times of the seven proposed heuristics can be seen in Table 1.4. For small sized scenarios, the Heuristics have very similar run-times with the exception of MaxMin which was much higher. The complexity of ObFun can not be seen in the average run-times for small scale problems, but for the



(a) Energy consumption for 100,000 tasks



(b) Makespan for 100,000 tasks

Figure 1.11 100,000 task problem-size

Table 1.4 Average run-time in seconds

No. of tasks	1,000	10,000	100,000
ObFun	$2.89E^{-2}$	0.439	21.16
Greedy-Min	$2.43E^{-2}$	0.233	2.38
Greedy-Deadline	$2.38E^{-2}$	0.228	2.34
Greedy-Max	$2.50E^{-2}$	0.235	2.47
MaxMin	$4.12E^{-2}$	0.404	4.04
MinMin StdDev	$2.42E^{-2}$	.262	2.49
MinMax StdDev	$2.42E^{-2}$	.252	2.45

large scale (10,000 and 100,000 tasks) it can be observed that ObFun takes considerably longer to execute than the rest of the Heuristics. The run-time of ObFun at 100,000 tasks is many times longer than the other Heuristics. It is this extended run-time that allows the objective function in ObFun to work and produce the dominating results seen in a 100,000 task problem.

To summarize, when solving problems with 1,000 tasks, Greedy-Min, Greedy-Deadline, Greedy-Max, MinMin StdDev and MinMax StdDev obtained solutions with the lowest energy consumption and the shortest *makespan*. For 10,000 task problems, ObFun, Greedy-Min, Greedy-Deadline, Greedy-Max, MinMin StdDev and MinMax StdDev demonstrated the highest solution quality. Finally, for the 100,000 task problems, ObFun vastly out performed all other heuristics. Overall, we may conclude ObFun is the best heuristic for large-sized problems. Greedy-Min, Greedy-Deadline, Greedy-Max, MinMin StdDev and MinMax StdDev can each outperform the rest depending on the heterogeneity of tasks and machines in the distributed system. Some examples were shown above to detail when a Heuristic will outperform the rest. Task and machine heterogeneity should be taken into account when choosing a Heuristic to model a distributed system.

1.5 RELATED WORKS

In this section we will discuss the related work to the proposed research. To keep the discussion short and relevant, only a subset of the related works will be discussed. This is due to the fact that a bulk of the related work has already been disseminated in surveys, such as [6], [30], and [31].

Most DPM techniques utilize instantaneous power management features supported by hardware. For example, in [1], 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 [5]. The DVS technique combined with a turn on/off technique is used to achieve high-power savings while maintaining all deadlines in [10]. In [27], 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 [25], [12], [13], [14], [21], genetic algorithms [8], [28], [9], [11], and constructive algorithms [7]. 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.

1.6 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. Seven heuristics were proposed to solve this problem. All seven of these heuristics were greedy heuristics, namely ObFun, Greedy-Min, Greedy-Deadline, Greedy-Max, MaxMin, MinMin StdDev, and MinMax StdDev. The seven heuristics were compared against each other with both small and large problem sizes. The simulation results showed that for small-sized problems, Greedy-Min, Greedy-Deadline, Greedy-Max, MinMin StdDev and MinMax StdDev 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.

REFERENCES

1. Tarek F. Abdelzaher and Chenyang Lu. Schedulability analysis and utilization bounds for highly scalable real-time services. In *In IEEE Real-Time Technology and Applications Symposium*, pages 15–25, 2001.
2. Ishfaq Ahmad, S. Ranka, and Samee Ullah Khan. 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*, pages 1–6, 2008.
3. Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Sahra Ali, and Debra Hensgen. Task execution time modeling for heterogeneous computing systems. In *HCW '00: Proceedings of the 9th Heterogeneous Computing Workshop*, page 185, Washington, DC, USA, 2000. IEEE Computer Society.
4. Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra Hensgen, and Sahra Ali. Representing task and machine heterogeneities for heterogeneous computing systems. *Tamkang Journal of Science and Engineering*, 3(3):195–207, 2000.
5. Ricardo Bianchini and Ram Rajamony. Power and energy management for server systems. *IEEE Computer*, 37(11):68–74, 2004.

6. Wissam Chedid and Chansu Yu. Survey on power management techniques for energy efficient computer systems. 2002.

7. Bharat P. Dave, Ganesh Lakshminarayana, and Niraj K. Jha. Cosyn: Hardware-software co-synthesis of embedded systems. In *DAC '97: Proceedings of the 34th annual Design Automation Conference*, pages 703–708, New York, NY, USA, 1997. ACM.

8. Robert P. Dick and Niraj K. Jha. Mogac: A multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17:920–935, 1997.

9. Robert P. Dick and Niraj K. Jha. Cowls: Hardware-software cosynthesis of wireless low-power distributed embedded client-server systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(1):2–16, 2004.

10. E.N. (Mootaz) Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *In Proceedings of the 2nd Workshop on Power-Aware Computing Systems*, pages 179–196, 2002.

11. Mohammad M. Hassani and Reza Berangi. Improving the COWLS algorithm for hardware sofware co-synthesis of wireless client-server systems using preference vectors and peak power information. In *Proc. Int'l Conference on Computer Systems and Technologies*, pages 1–5, 2007.

12. Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *PPoPP '05: Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 186–195, New York, NY, USA, 2005. ACM.

13. Chung Hsing Hsu and Ulrich Kremer. The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. In *In Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 38–48. ACM Press, 2003.

14. Chi-Hong Hwang and Allen C.-H. Wu. A predictive system shutdown method for energy saving of event-driven computation. In *1997 Design Automation Conference*, pages 28–32, 1997.

15. Samee U. Khan and Ishfaq Ahmad. A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):346–360, 2009.

16. Samee Ullah Khan. 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.

17. Samee Ullah Khan and Ishfaq Ahmad. Comparison and analysis of ten static heuristics-based internet data replication techniques. *Journal of Parallel and Distributed Computing*, 68(2):113–136, 2008.

18. Samee Ullah Khan and Ishfaq Ahmad. A cooperative game theoretical replica placement technique. In *2007 International Conference on Parallel and Distributed Systems*, 200**7**.

19. Samee Ullah Khan and C. Ardil. On the joint optimization of performance and power consumption in data centers. In *International Conference on Distributed, High-Performance and Grid Computing*, pages 660–660, 2009.

20. Samee Ullah Khan and Cemal Ardil. Energy efficient resource allocation in distributed computing systems. In *International Conference on Distributed, High-Performance and Grid Computing*, pages 667–673, 2009.

21. Darko Kirovski and Miodrag Potkonjak. System-level synthesis of low-power hard real-time systems. In *DAC '97: Proceedings of the 34th annual Design Automation Conference*, pages 697–702, New York, NY, USA, 1997. ACM.

22. Yan Alexander Li, John K. Antonio, Howard Jay Siegel, Min Tan, and Daniel W. Watson. Determining the execution time distribution for a data parallel program in a heterogeneous computing environment. *Journal of Parallel and Distributed Computing*, 44(1):35–52, 1997.

23. David. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, 1984.

24. Cleve B. Moler. *Numerical Computing with Matlab.* Society for Industrial Mathematics, 2004.

25. Ripal Nathuji, Canturk Isci, and Eugene Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, 2007.

26. Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, 1984.

27. Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *In Workshop on Compilers and Operating Systems for Low Power*, 2001.

28. Marcus T. Schmitz and Bashir M. Al-Hashimi. Considering power variations of dvs processing elements for energy minimisation in distributed systems. *Proc. 14th Int'l Symp. on System Synthesis*, pages 250–255, 2001.

29. Howard Jay Siegel and Shoukat Ali. Techniques for mapping tasks to machines in heterogeneous computing systems. *Journal of Systems Architecture*, 46(8):627–639, 2000.

30. Osman S. Unsal and Israel Koren. System-level power-aware design techniques in real-time systems. In *Proceedings of the IEEE*, pages 1055–1069, 2003.

31. Vasanth Venkatachalam and Michael Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys*, 37(3):195–237, 2005.

32. Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *OSDI '94: Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, pages 13–23, Berkeley, CA, USA, 1994. USENIX Association.

33. Yang Yu and Viktor K. Prasanna. 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*, pages 341–348, Washington, DC, USA, 2002. IEEE Computer Society.