J Supercomput
DOI 10.1007/s11227-011-0611-7

A goal programming based energy efficient resource allocation in data centers

Samee Ullah Khan · Nasro Min-Allah

Abstract We study the multi-objective problem of mapping independent tasks onto a set of data center machines that simultaneously minimizes the energy consumption and response time (makespan) subject to the constraints of deadlines and architectural requirements. We propose an algorithm based on goal programming that effectively converges to the compromised Pareto optimal solution. Compared to other traditional multi-objective optimization techniques that require identification of the Pareto frontier, goal programming directly converges to the compromised solution. Such a property makes goal programming a very efficient multi-objective optimization technique. Moreover, simulation results show that the proposed technique achieves superior performance compared to the greedy and linear relaxation heuristics, and competitive performance relative to the optimal solution implemented in Linear Interactive and Discrete Optimizer (LINDO) for small-scale problems.

Keywords Data center · Multi-objective optimization · Goal programming · Resource allocation

1 Introduction

In today's computerized world, industries, regardless of their domain are highly dependent on critical computer applications. These applications are usually hosted in

S.U. Khan (✉)
Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108, USA
e-mail: samee.khan@ndsu.edu

N. Min-Allah
Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, Pakistan
e-mail: nasar@comsats.edu.pk

Published online: 19 April 2011

dedicated servers that are kept in large facilities called data centers. The Internet, in particular, has spawned an Information Technology (IT)-intensive e-commerce industry that relies heavily on data centers. A recent report on the state of today's data centers by Symantec Corporation lists complexity as the major issue in managing data centers [1]. Many competing factors call for attention. Delivering IT services in a reliable and timely manner, often referred to as response time, is the very *raisons dêtre* of a data center. Energy consumption, on the other hand, also has gained a lot of attention in the last decade. Understanding these trade-offs between such competing factors is the key to managing a data center well.

Energy-efficient techniques for managing a system at runtime can bring down the amount of energy it consumes. These management techniques are mostly aimed at:

- reducing the energy wasted by transitioning a system to its sleep mode when it is idle and
- reducing the energy consumed by slowing down the system during lean (but not idle) periods.

The former technique is called Dynamic Power Management (DPM) [13], while the latter is called Dynamic Voltage Scaling (DVS) [21] (or Speed Scaling in the more theoretical literature [4]).

DPM considers a system (in the simplest case a processor) that can be in one of the two states that we call the active state and the sleep state. The system can handle requests only in its active state, but the active state consumes far more energy per unit time compared to the sleep state. However, when a request arrives while the system is in the sleep state, it must "wake up" and assume the active state before the request can be served. This transition from sleep to active state has a high transition cost, and is not a favorable approach undertaken by researches and vendors [6].

DVS on the other hand, seeks to exploit the convex relationship between the Central Processing Unit (CPU) supply voltage (that impacts the speed of execution) and the power consumption. The power consumption in Complementary Metal Oxide Semiconductor (CMOS) circuits is given by $P = V^2 \times f \times C_{\text{EFF}}$, where V, f, and C_{EFF} are the supply voltage, clock frequency, and effective switched capacitance of the circuits, respectively. Moreover, we also know that the time to finish an operation is inversely proportional to the frequency. Furthermore, power is the rate at which energy is consumed. Therefore, the energy per operation is proportional to V^2, which implies that lowering the supply voltage quadratically decreases the energy. However, lowering the supply voltage reduces the maximum allowable clock speed (or frequency) in an approximately linear manner. This leads us to the cube rule in CMOS circuits which states that the instantaneous power is roughly proportional to the clock speed cubed. The main objective, therefore, is to keep the supply voltage (or clock speed) as low as possible so that the power consumption is minimal, but without compromising Quality of Service (QoS) measures [32]. In this paper, we will investigate the joint optimization of energy consumption and response time. Because response time improves when makespan improves, we must use makespan as the primary criteria to determine improvement in response time. Moreover, because power is simply the rate at which energy is consumed, we must optimize the instantaneous power.

Our contributions: This paper is principally concerned with designing an efficient and effective data center resource allocation methodology.

1. A problem is formulated to minimize the instantaneous power (and consequently the energy consumption) and the response time (makespan) while maintaining performance requirements, such as deadline constraints and tasks' architectural requirements.
2. We propose a high solution quality technique based on the goal programming methodology.
3. The proposed goal programming approach also is compared against the optimal solution implemented in Linear Interactive and Discrete Optimizer (LINDO) [28], the greedy [33] and linear relaxation [33] heuristics. The simulation results reveal the superior performance of the proposed technique compared to other traditional methodologies.

The remainder of this paper is organized as following. The system model and problem formulation are discussed in Sect. 2. Section 3 provides some essential information pertaining to goal programming and details our proposed approach. Simulation results and related work are provided in Sects. 4 and 5, respectively. Finally, in Sect. 6, we summarize our investigation.

2 System model and problem description

2.1 The system model

Consider a data center comprising of a set of machines, $M = \{m_1, m_2, \ldots, m_m\}$. Assume that each machine is equipped with a DVS module and is characterized by:

1. The frequency of the CPU, f_j, given in cycles per unit time. With the help of a DVS, f_j can vary from $f_j^{\min}$ to $f_j^{\max}$, where $0 < f_j^{\min} < f_j^{\max}$. From frequency, it is easy to obtain the speed of the CPU S_j that is approximately proportional to the frequency of the machine [23, 33].
2. The specific machine architecture, $A(m_j)$. The architecture would include the type of CPU, bus types, and speeds in GHz, I/O, and memory in bytes.

Consider a metatask, $T = \{t_1, t_2, \ldots, t_n\}$. Each task is characterized by:

1. The computational cycles c_i that it needs to complete. The assumption here is that the c_i is known *a priori*.
2. The specific machine architecture $A(t_i)$ that it needs to complete its execution.
3. The deadline, d_i, before it has to complete its execution. Moreover, we also assume that the metatask, T, also has a deadline D that is met if and only if the deadlines of all its tasks are met.

The number of computational cycles required by t_i to execute on m_j is assumed to be a finite positive number, denoted by c_{ij}. The execution time of t_i under a constant speed S_{ij}, given in cycles per second is $t_{ij}/c_{ij} = S_{ij}$. For the associated data and instructions of a task, we assume that the processor always retrieves it from the level-1 (primary) data cache. A task, t_i, when executed on machine m_j draws, p_{ij} amount of instantaneous power. Lowering the instantaneous power will lower the CPU frequency, and consequently will decrease the speed of the CPU and hence cause t_i to possibly miss its deadline.

The architectural requirements of each task are recorded as a tuple with each element bearing a specific requirement. We assume that the mapping of architectural requirements is a Boolean operation. That is, the architectural mapping is only fulfilled when all of the architectural constraints are satisfied, otherwise not.

2.2 Problem formulating

Find the task to machine mapping, where the cumulative instantaneous power consumed by the data center, M and the makespan of the metatask, T, is minimized.

Mathematically, we can say

$$\text{minimize}\left(\sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}x_{ij} \text{ and } \max_{j}\sum_{i=1}^{n} t_{ij}x_{ij}\right) \tag{1}$$

$$\text{subject to } x_{ij} \in \{0, 1\}, \tag{2}$$

$$t_i \rightarrow m_j; \quad \text{if } A(t_i) = A(m_j) \quad \text{then } x_{ij} = 1, \tag{3}$$

$$t_{ij}x_{ij} \leq d_i | x_{ij} = 1, \tag{4}$$

$$(t_{ij}x_{ij} \leq d_i) \in \{0, 1\}, \tag{5}$$

$$\prod_{i=1}^{n} (t_{ij}x_{ij} \leq d_i) = 1 | x_{ij} = 1. \tag{6}$$

Constraint (2) is the mapping constraint. When $x_{ij} = 1$, a task, t_i, is mapped to machine, m_j, and $x_{ij} = 0$ otherwise. Constraint (3) elaborates on this mapping in conjunction to the architectural requirements, and it states that a mapping can only exist if the architecture is mapped. Constraint (4) relates to the fulfillment of the deadline of each task, and constraint (5) tells us about the Boolean relationship between the deadline and the actual time of execution of the tasks. Constraint (6) relates to the deadline constraints of the metatask that will hold if all of the deadlines of the tasks, d_i, are satisfied.

The aforementioned problem formulation is in a form of multi-objective optimization problem. In the literature, there are two standard ways to tackle such multi-objective problems: (a) optimize objectives concurrently or (b) optimize one objective first, then make that as a constraint for the rest of the objectives.

To optimize one objective first, then make that as a constraint for the other objectives, the only plausible framework is when one can ensure that the objective functions have an acceptable overlap [12]. Because of the multi-objective nature of the problem described in this paper, we must choose to optimize both the objectives concurrently.

3 Goal programming

3.1 Background information

In uni-objective problems, the aim of an optimization technique is the identification of the optimal solution [20]. That is to say the identification of the feasible solution(s) that gives the best value of the objective function. The stringent notion of optimality must be dropped when considering multi-objective problems. In general, a solution that optimizes one objective function may not optimize other objective functions [19].

In goal programming, the decision maker indicates the goals that are the desired achievement levels of the objective functions [34]. In certain cases, the decision maker may also provide the relevant rankings of the achievement levels. This is a tedious process because a high-level human intervention must be a part of the whole solution process. High-level intervention may be error-prone, over simplified, or at times very complicated [20]. Therefore, an algorithmically determinable achievement level must be conceived.

The choice of goal programming also is important for large-scale problems because it is a very efficient technique to resolve fluctuations in solution quality given a certain set of input parameters [34]. This is one of the primary reasons that we use such a technique to solve the energy-efficient scheduling problem in this paper. In addition to developing optimality conditions for uni-objective problems, Kuhn and Tucker also stated conditions for non-inferiority in multi-objective problems [22]. A feasible solution is non-inferior if there exists no other feasible solution that will yield an improvement in one objective without causing a degradation in at least one other objective. These conditions may serve as achievement levels of the objective functions. This will be the topic of Sect. 3.3, but first we detail a generalized goal programming approach in the subsequent section.

3.2 A generalized goal programming procedure

Goal programming implicitly assumes that a desired goal is always obtainable and that it can be used to converge to a compromised solution through some high-level intervention [34]. The information attainable during each of the iteration is the current best *compromise* solution, referred to as the Main Solution (MS), and a set of Possible Solutions (PS) that are the compromise solutions obtainable if each of the goals are satisfied serially [19]. Iteratively, goal programming, identifies non-inferior solutions and refines them to achieve the best possible compromise solution. An iteration can be classified as a two-step calculation and evaluation process. During the calculation step, the MS and PS are obtained that are analyzed to proceed toward a compromise solution during the evaluation process. During the iterative process, if the evaluation procedure determines that either the MS or the one of the PS is the best compromise solution, then the procedure terminates [20]. Below we describe a generalized procedure for goal programming.

Let the multi-objective problem to be solved be

$$\min\big(f_1(x), f_2(x), \ldots, f_k(x)\big), \tag{7}$$

$$\text{such that } g_j(x) \leq 0, \quad i = 1, 2, \ldots, m,$$

where x is an n dimensional decision variable vector. The following steps must be present for a generalized goal programming approach.

Step 0: Determine f_i^* and f_{*i}, as

1. $\min f_i(x)$ such that $g(x) \leq 0$.
   The solution to the above, referred to as x_{*i} and f_{*i} is known as the *ideal solution* [20]. Let $f_{ij} = f_i(x)$, then
2. $f_i^* = \max_j f_{ji}$.

The functions f_i^* and f_{*i} provide the upper and lower bounds on the objective function values, respectively. Such values are important to guide the solution toward the desirable compromise solution. Moreover, they also determine the feasibility of the solution space.

Step 1: Set initial goals, $b = \{b_1, b_2, \ldots, b_k\}$. As mentioned previously a high-level intervention must determine a desirable goal for each and every objective function. However, one can approximate these goals by determine the necessary and sufficient conditions of optimality—the Kuhn–Tucker conditions. It should be clear that $f_{*i} < b_i \leq f_i^*$.

Step 2: Solve for MS.

$$\min a = \left(\left(\sum_i d_i^-\right), \left(\sum_i -d_i^+\right)\right), \tag{8}$$

$$\begin{gathered}
\text{such that } g(x) \leq 0, \\
f_i(x) + w_i \times d_i^- - w_i \times d_i^+ = b_i, \\
d_i^- \times d_i^+ = 0, \\
d_i^- \leq 1, \\
d_i^-, d_i^+ \geq 0,
\end{gathered} \tag{9}$$

where $w_i = b_i - f_{*i}$. The optimization of the MS would result in x^0 and f^0, which is known in the literature as the core solution [20]. The weight w has been derived from a normalizing scheme that makes the variation between f_{*i} and b_i equal to one. That is,

$$\frac{f_i(x) - f_{*i}}{b_i - f_{*i}} + d_i^- - d_i^* = \frac{b_i - f_{*i}}{b_i - f_{*i}} = 1, \tag{10}$$

where $f_{*i} < b_i \leq f_i^*$; hence, we obtain the following:

$$f_i(x) + (b_i - f_{*i})\, d_i^- - (b_i - f_{*i})\, d_i^+ = b_i, \tag{11}$$

which is the same as (9) after substituting $w_i = b_i - f_{*i}$. Moreover, the constraint $d_i^- \leq 1$ ensures that $f_i(x)$ does not violate the lower bound, f_{*i}. Furthermore, the weights w have the following two additional properties: (a) the value of the weight is dynamically adjusted as the value of a goal alters in between an iteration, and (b) the weight increases whence the value of goal decreases and vice versa.

Step 3: Solve for PS.

$$\min a_r = \left(\left(\sum_{i,r \neq i} d_i^- \right), \left(\sum_i -d_i^+ \right) \right), \tag{12}$$

$$\text{such that } g(x) \leq 0,$$

$$f_i(x) - w_i \times d_i^+ = b_i,$$

$$f_i(x) + w_i \times d_i^- - w_i \times d_i^+ = b_i,$$

$$d_i^- \times d_i^+ = 0,$$

$$d_i^- \leq 1,$$

$$d_i^-, d_i^+ \geq 0,$$

where $w_i = b_i - f_{*i}$. The optimization of the PS would result in x^r and f^r, which is known in the literature as the *achievable solution* [34].

The goal programming approach must iterate between Steps **2** and **3** to arrive at a compromised solution. The question that how does one obtain a proper weight for a given optimization problem is the topic of the subsequent section. Moreover, by deriving the necessary and sufficient conditions of optimality, one ensures that the optimization process is convergent. Furthermore, the solution space is reduced only to Pareto frontier [9].

3.3 Conditions of optimality

Because we must have an upper bound on the power consumption, we introduce power conservation conditions to the set of constraints (2), (3), (4), (5), and (6).

$$\sum_{i=1}^{n} \sum_{m=1}^{m} p_{ij} \leq P, \tag{13}$$

$$p_{ij} \geq 0, \quad i = 1, 2, \ldots, n; \; j = 1, 2, \ldots, m. \tag{14}$$

The power conservation condition of (13) states that the instantaneous power allocated is bounded. That is, at any given instance, the total power consumption of all of the machines must be less than when all of the machines are running at their peak

power consumption. Clearly, the instantaneous power consumption must be a positive number, as in (14). These constraints make the multi-objective problem convex that in turn makes the optimization problem tractable [29]. Moreover, because the instantaneous power regulates the time to complete a task, it is sufficient to utilize power as the only tunable variable to derive the conditions of optimality. Let $\alpha \leq 0$ and $\eta_j \leq 0$ denote the Lagrange multipliers, and γ_j be the gradient of the binding constraints [22]. Then we can say that the Lagrangian is

$$L(\gamma_j, \alpha, \eta_j) = \sum_{i=1}^{n}\sum_{j=1}^{m} \ln\left(\gamma_j(\alpha) - p_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{m}(\gamma_j - P)\right) + \sum_{i=1}^{n}\sum_{j=1}^{m} \eta_j\left(\gamma_j - p_{ij}\right).$$

The first-order Kuhn–Tucker conditions are given in (15) and (16), with a constraint given in (17):

$$\frac{\delta L}{\delta \gamma_j} = \frac{-1}{\gamma_j - p_{ij}} + \alpha\eta_j = 0, \tag{15}$$

$$\frac{\delta L}{\delta \alpha} = \sum_{j=1}^{m} \gamma_j - P = 0, \tag{16}$$

$$\gamma_j - p_{ij} \geq 0, \quad \eta_j(\gamma_j - p_{ij}) = 0, \quad \eta_j \leq 0. \tag{17}$$

If $\gamma_j - p_{ij} = 0$, then the current instantaneous power consumption is the best instantaneous power. If $\gamma_j - p_{ij} > 0$, then $\eta_j = 0$. The solution (or the derivative) of (15) and (16) is given in (18) and (19), respectively.

$$\frac{1}{\gamma_j - p_{ij}} - \alpha = 0, \tag{18}$$

$$\sum_{j=1}^{m} \gamma_j = P. \tag{19}$$

It then follows that

$$\gamma_j = \frac{P - \sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}}{m}. \tag{20}$$

Because γ_j by definition is the gradient of the binding constraints, we can replace γ_j with p_{ij}. That gives us

$$p_{ij} = \frac{P - \sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}}{m}. \tag{21}$$

Now, for a specific machine j, the optimality must oscillate between the instantaneous power consumed by machine j and the rest of $m-1$ machines. Therefore, the

following must hold:

$$p_{ij} = \frac{P - \sum_{i=1}^{n} \sum_{\forall\, k \in M, k \neq j} p_{ik}}{m}. \tag{22}$$

The Kuhn–Tucker conditions verify the following: (a) The non-inferior solutions form the Pareto frontier when the instantaneous power consumption of a machine j (that has mapped a task) is below the peak power consumption of machine j. (b) The goal is achieved whence machine j is operating on an instantaneous power that is scaled as the mth lowest power consumption of machine j. It also is worth reporting that due to the linearity relationship between power consumption and the associated task completion time, the conditions of optimality are sufficient to consider only one single constraint—instantaneous power. Utilizing both of the constraints would have resulted in a similar conditions of optimality; however, the derivation would have been complicated. In the next section, we will outline our goal programming based task to machine mapping technique.

3.4 Goal programming based technique (GP)

We have all the necessary components to propose a goal programming based task to machine mapping technique, acronym GP. The GP technique takes in as an input the sets M and T with all of the machines initialized to their corresponding highest level of DVS.

To derive an upper and lower bound on the desired goal (corresponding to Step **0** of Sect. 3.2), we must utilize the earliest deadline first approach. This will ensure that the classical claim by the earliest deadline first approach is satisfied. That is, if T can be scheduled (by an optimal algorithm) such that constraint (4) is satisfied, then the earliest deadline first approach will schedule T such that constraint (6) is satisfied. The earliest deadline first approach also will ensure that the GP technique has a corresponding upper and lower bound on instantaneous power consumption and makespan given deadlines of the metatask. The corresponding bounds will be dictated by the tightness of the associated deadlines. That is, the tighter the deadline for a given task, the more the instantaneous power would be consumed, and vice versa.

The Kuhn–Tucker conditions derived in Sect. 3.3 set the initial goals corresponding to Step **1** of Sect. 3.2. They are not depicted in Algorithm 1 that describes the GP technique. Instead, implicitly, Steps **2** and **3** guide the solution toward a best possible compromise [31].

To develop a MS (corresponding to Step **2** of Sect. 3.2.), we must satisfy constraint (3). First, we limit our solution space to only those machines, $\mathcal{M}$, that can satisfy the architectural constraint. To ensure a feasible MS, we must identify machines that without altering their current DVS level can finish the task within the specified deadline. Such an assurance is also known as laxity [18]. A laxity set, Δ , is constructed. Using Δ, we determine the best possible task to machine mapping without any alteration to the DVS levels. This is accomplished by picking the machine that exhibits the minimum laxity. The MS is not complete until an optimum level of

Input: T and M.
Output: Task to machine mapping.
Initialize: $\forall j$, DVS_j is set to the highest level.
while $T \neq \emptyset$ **do**
  $\mathcal{I} \leftarrow \text{argmin}_i(d_i)$ **foreach** $m_j \in M$ **do**
    if $A(t_{\mathcal{I}}) = A(m_j)$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \cup m_j$;
    end
    if $\mathcal{M} = \emptyset$ **then**
      EXIT;
    end
  end
  foreach $m_j \in \mathcal{M}$ **do**
    $\Delta_{\mathcal{I}j} \leftarrow d_{\mathcal{I}} - t_{\mathcal{I}j}$;
    if $\Delta_{\mathcal{I}j} > 0$ **then**
      $\Delta \leftarrow \Delta \cup \Delta_{\mathcal{I}j}$;
    end
  end
  if $\Delta = \emptyset$ **then**
    foreach $m_j \in \mathcal{M}$ **do**
      Reset DVS_j to the highest level;
      $\Delta_{\mathcal{I}j} \leftarrow d_{\mathcal{I}} - t_{\mathcal{I}j}$;
      if $\Delta_{\mathcal{I}j} > 0$ **then**
        $\Delta \leftarrow \Delta \cup \Delta_{\mathcal{I}j}$;
      end
      if $\Delta = \emptyset$ **then**
        EXIT;
      end
    end
  end
  $\mathcal{J} \leftarrow \text{argmin}_j(\Delta)$;
  while $\{t_{\mathcal{I}\mathcal{J}} x_{\mathcal{I}\mathcal{J}} \leq d_{\mathcal{I}} | x_{\mathcal{I}\mathcal{J}} = 1\}$ **do**
    Reduce $\text{DVS}_{\mathcal{J}}$ by one level;
  end
  $i \leftarrow \mathcal{I}$;
  $j \leftarrow \mathcal{J}$;
  $x_{ij} \leftarrow 1$;
  $T \leftarrow T - \{t_i\}$;
end

Algorithm 1: The goal programming based task to machine mapping technique (GP)

DVS is determined. The DVS level of the chosen machine j is lowered until constraint (4) is violated. This ensures that the mapped task is running on a machine that can fulfill all of the constraints and consuming an instantaneous power that results in a compromised solution.

The MS will be stable as long as Δ can be constructed. However, mapped tasks stack-up on machines, thereby reducing the laxity, and possibly to a level that Δ

is empty. Once that happens PS must be constructed corresponding to Step **3** of Sect. 3.2. Because only set $\mathcal{M}$ can potentially satisfy constraint (4), the PS must be from within $\mathcal{M}$. To increase laxity, the machines must operate on their corresponding highest speed levels (or highest DVS levels), respectively. These new DVS levels will pad (down) the stacked tasks on machines to levels that can ensure that we have one feasible (positive) laxity. (The pad down is achieved by running all the mapped tasks on the highest speed. This will lower the makespan; hence ensuring a feasible laxity.) Set Δ is reconstructed and the machine that ensures the minimum laxity is chosen as the PS.

The GP heuristic as mandated in Sect. 3.2 oscillates between MS and PS. A number of important conclusions also can be deduced from the GP technique. Namely,

1. A feasible solution if it exists is always identified; otherwise the **EXIT** statements identify unfeasible solutions based on constraint (3) or the laxity criterion.
2. If the algorithm maps all of the tasks on machines within the MS construction, then the solution is optimal. Moreover, deadlines must be very loose. Furthermore, the laxity must be very high.
3. If the algorithm constructs PS, then the solution is on the Pareto frontier (definition of Kuhn–Tucker conditions of Sect. 3.3). Moreover, PS ensure that optimum solution is identified (definition of PS, Sect. 3.2). Furthermore, PS revisits MS to rectify anomalies by altering the corresponding DVS levels such that the resultant is a feasible optimal compromise.

Finally, to ensure that the GP technique is tractable, we analyze the termination time. It is easy to congregate that the exact worst-case bound is $\mathcal{O}(n^2 \log n + 3mn + mn \log m)$. Because it is assumed that $m \ll n$, the worst-case bound reduces to $\mathcal{O}(n^2 \log n)$.

4 Simulations, results, and discussion

We set forth two major goals for our simulation study: (a) To measure and compare the performance of the proposed technique against the optimal solution, greedy heuristic [33], and linear relaxation (LR) heuristic [33]. (b) To measure the impact of system parameter variations. We choose to compare the proposed technique against the above mentioned two heuristics because they have shown to perform extremely well compared to several other heuristics [33]. Due to space restrictions, we could not include the finer details of the greedy and LR heuristics. However, we strongly encourage the readers to review the referenced articles.

Based on the size of the problems, the simulations were divided in two parts. For small-size problems, we used an Integer Linear Programming toolkit called LINDO [28]. LINDO is useful to obtain optimal solutions, provided that the problem size is relatively small. Hence, for small problem sizes, the relative performance of the proposed technique, greedy, and LR techniques is compared against the LINDO implementation. For large-size problems, it is impractical to compute the optimal solution. Hence, we consider comparisons only against the greedy and LR heuristics.

Table 1 Problem characteristics for the 12 days of AFSCN data used in our simulations

Date	No. of requests	High	Low
10/12/92	322	169	153
10/13/92	302	165	137
10/14/92	311	165	146
10/15/92	318	176	142
10/16/92	305	163	142
10/17/92	299	155	144
10/18/92	297	155	142
03/07/02	483	258	225
03/20/02	457	263	194
03/26/03	426	243	183
04/02/03	431	246	185
05/02/03	419	241	178

For the workload, we acquired the data from the US Air Force Satellite Control Network (AFSCN). The data is publicly available over the Internet at [2]. The AFSCN is currently responsible for coordinating communications between civilian and military organization and more than 100 United States Air Force (USAF) managed satellites. The satellite-ground communications are performed using sixteen antennas located at nine tracking stations around the globe. Table 1 summarizes the characteristics of the data.

There are two types of task requests that can be distinguished: (a) low-altitude and (b) high-altitude orbits. The low-altitude tasks specify requests for low-altitude satellites; such requests tend to be very short (on the average they are 5–10 minutes in duration) and have a tight visibility window (simply because the relative velocity of low-altitude satellites is very high compared to high-altitude satellites). High-altitude tasks specify requests for high-altitude satellites; the durations for these requests are more varied and usually longer, with large visibility windows. A problem instance of AFSCN consists of n task requests. Each task request t_i, $1 \leq i \leq n$, specifies a required processing duration (i.e., the time to complete a task) and the associated data file (i.e., the amount of memory required to process a task). Each task request also specifies a number of pairs of the form (t_i, m_j), each identifying a particular alternative resource (i.e., an antenna or, in our context, a machine m_j) and time window, i.e., the deadline for a task d_i. The deadlines for low-altitude tasks were relatively tighter than high altitude tasks. The processing duration of the task is the same for all possible alternative resources, and it needs to be mapped to a resource and completed within the time window.

Finally, to model the DVS modules, we make the generic assumption that the task processing times were given when machines were running at full instantaneous power (or at a level of DVS equal to 100%). Because the task processing time, as given in the AFSCN data, is uniform across all machines, for this problem instance, we assume that the machines have a clock speed of two GHz. We also assume that

a potential difference of one mV across a CMOS circuit generates a frequency of one MHz. Altering any of these assumptions will be a trivial task and will have no significant impact on the simulation results. For this study, we keep the architectural affinity requirements confined to memory. Adding other requirements, such as I/O and processor type, will bear no affect on our simulation setup. In all of the simulation setups, the storage capacity of the machines was set proportional to the total size of data items (TS). The capacity of a machine was generated using a uniform distribution from ($0.5 \times TS$) and ($1.5 \times TS$). For small-size problems, the number of machines was fixed at five, while the number of tasks varied from five to 50. The tasks were chosen on random from each of the twelve days of the AFSCN workload and the simulation plot is an average over the twelve runs. The number of DVS levels per machine was set to four. For large-size problems, the number of machines was fixed at sixteen, while the number of tasks varied from 322 to 4 370. The bound of the number of tasks reflect the sequential aggregation of the tasks in the AFSN workload, i.e., for the first set, we use the data from 10/12/92 having 322 tasks, for the second set, we use the data as the aggregate of the data of 10/12/92 (322 tasks) and 10/13/92 (302 tasks), giving us a total of 624 tasks, and so on. The number of DVS levels per m_j was set to sixteen.

The simulation results for small size problems with are reported in Figs. 1 and 2. These figures show the ratio of the makespan obtained from the three techniques and the optimal. The plot in Fig. 1 shows that the GP (proposed) technique performs extremely well by achieving a makespan within 9% of the optimal solution. Next, we compare the overall energy consumption that is calculated as the time interval a task takes to complete on a given machine multiplied by the current instantaneous power of the given machine. In Fig. 2, we observe that once again, the GP technique outperforms the other techniques in terms of energy savings compared to the optimal allocation within a range of seventeen percent.

For large problem instances, first, we compare the makespan identified by the GP, greedy, and LR heuristics. Figure 3 shows the performance of the techniques. The results indicate that proposed technique outperforms the greedy and LR heuristics in identifying a smaller makespan. We can observe that the GP technique identifies a makespan that is 24.03% smaller than greedy and 27.39% smaller than the LR heuristic. Second, we compare the energy consumption of the three techniques. Figure 4 shows the relative performance of the techniques. GP again outperforms the other heuristics by consuming lesser energy when executing the tasks. We observe that the GP technique saves on average 22.05% of energy than the greedy and 48.34% of energy than the LR heuristic.

Last, we analyze the runtime for both small and large problem sizes. For completion, the runtime of the optimal for small problem size is presented for comparisons. The results are depicted in Figs. 5 and 6. The GP technique terminates many orders faster than the optimal, and the greedy and LR heuristics. This suggests that the GP technique not only terminates faster but also delivers a superior solution quality than the compared techniques.

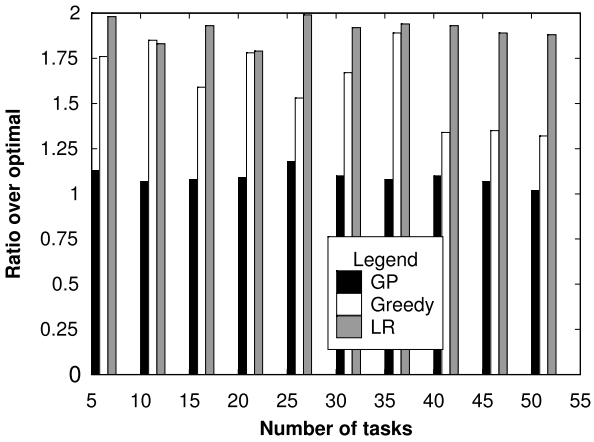

Fig. 1 Makespan ratio over the optimal

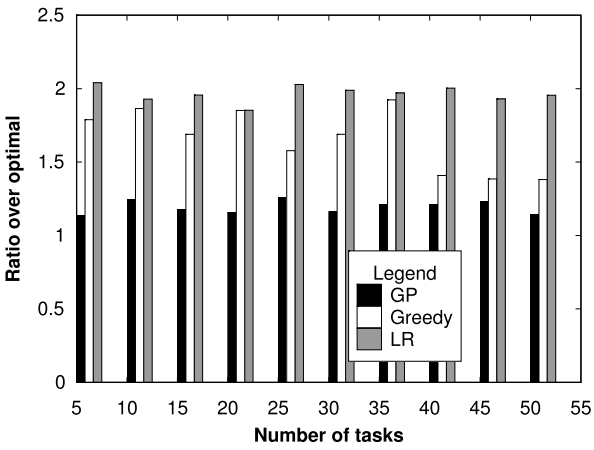

Fig. 2 Energy consumption ratio over the optimal

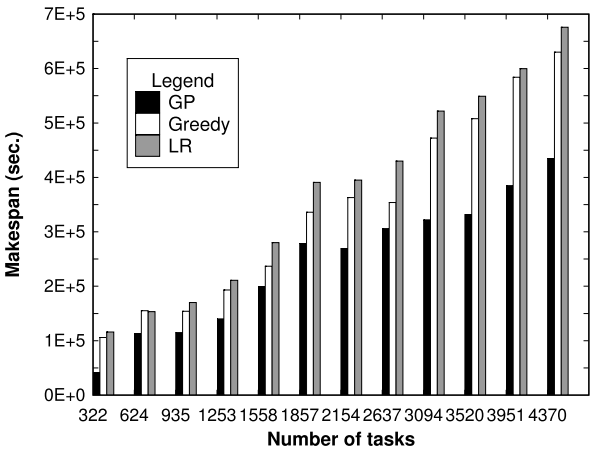

Fig. 3 Makespan

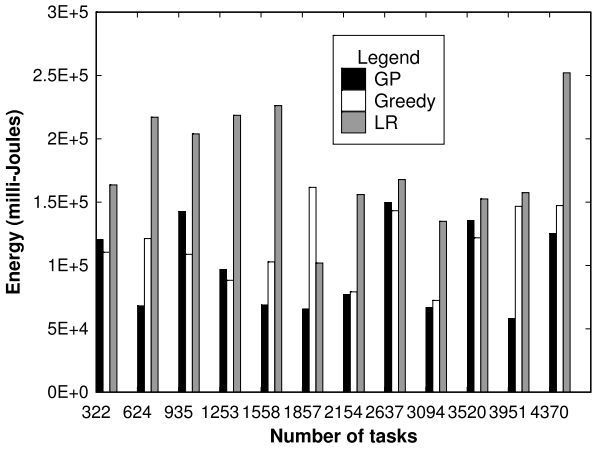

Fig. 4 Energy consumption

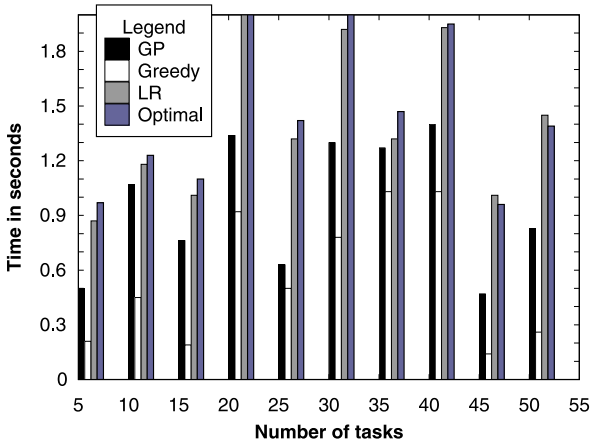

Fig. 5 Avg. execution time (small size problems)

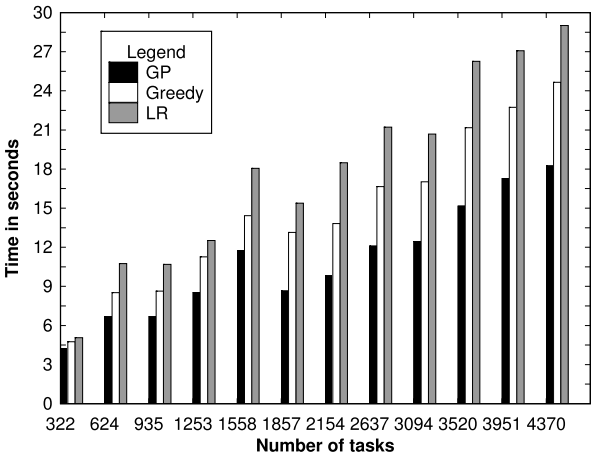

Fig. 6 Avg. execution time (large size problems)

5 Related work

Most DPM techniques utilize instantaneous power management features supported by hardware. For example, [3] extends the operating system's power manager by an adaptive power manager (APM) that uses the processor's DVS capabilities to decrease or increase the CPU frequency, thereby minimizing the overall energy consumption [5]. The DVS technique at the processor-level together with a turn on/off technique at the cluster-level to achieve high-power savings while maintaining the response time is proposed in [27]. In [26], the authors introduce a scheme to concentrate the workload on a limited number of servers in a cluster such that the rest of the servers can remain switched-off for a longer period of time.

In the recent years, there has been a surge in work on energy efficient scheduling. Most of the work has been in the context of grid computing and cloud computing. The authors are aware of works in the aforementioned domain that originated from fields, such as game theory, discrete optimization, heuristics, and linear optimization. A selection of the aforementioned can be found in the following papers [10, 14–17, 25, 30].

While the closest techniques to combining device power models to build a whole system has been presented in [7], our approach aims at building a general framework for autonomic power and performance management, where we bring together and exploit existing device power management techniques from a whole system's perspective. Furthermore, while most power management techniques are either heuristic-based approaches, such as [11] and [24] or stochastic optimization techniques, such as [8] and [32], we use goal programming to seek radically fast and efficient solutions compared to the traditional approaches.

We must understand that all of the aforementioned works have been designed for systems and tested on workloads that are considerably different from each other. However, the authors do take into notice that all of the aforementioned techniques do and can adapt to various systems and workloads with minor adjustments.

6 Conclusions

This paper presented an energy optimizing power-aware resource allocation strategy in data centers. A solution from goal programming was proposed for this multi-objective problem. The solution quality of the proposed technique was compared against the optimal for small-scale problems, and greedy and linear relaxation heuristics for large-scale problems. The simulation results confirm superior performance of the proposed scheme in terms of reduction in energy consumption and makespan compared to the heuristics and the optimal solution obtained using LINDO.

References

1. Symantec corporation. Symantec state of the data center report, available online at: http://www.symantec.com/content/en/us/about/media/SOTDC_report_2007.pdf

2. United States air force satellite control network data, available online at: http://www.cs.colostate.edu/sched/index.html
3. Abdelzaher TF, Lu C (2001) Schedulability analysis and utilization bounds for highly scalable real-time services. In: 7th real-time technology and applications symposium, p 15
4. Bansal N, Kimbrel T, Pruhs K (2004) Dynamic speed scaling to manage energy and temperature. In: 45th annual IEEE symposium on foundations of computer science, pp 520–529
5. Bianchini R, Rajamony R (2004) Power and energy management for server systems. IEEE Comput 37(11):68–74
6. Bunde DP (2006) Power-aware scheduling for makespan and flow. In: 8th ACM symposium on parallelism in algorithms and architectures, pp 190–196
7. Chen J, Dubois M, Stenström P (2007) Simwattch: Integrating complete-system and user-level performance and power simulators. IEEE MICRO 27(4):34–48
8. Chung E-Y, Benini L, Bogiolo A, De Micheli G (1999) Dynamic power management for non-stationary service requests. In: Conference on design, automation and test in Europe, p 18
9. Dyer JS (1972) Interactive goal programming. Oper Res 19:62–70
10. Guzek M, Pecero JE, Dorronsoro B, Bouvry P, Khan SU (2010) A cellular genetic algorithm for scheduling applications and energy-aware communication optimization. In: International conference on high performance computing & simulation HPCS, pp 241–248
11. Heath T, Diniz B, Carrera EV, Meira W Jr, Bianchini R (2005) Energy conservation in heterogeneous server clusters. In: 10th ACM SIGPLAN symposium on principles and practice of parallel programming, pp 186–195
12. Hwang CL, Masud ASM (1979) Multiple objective decision making—methods and applications: A state-pf-the-art survey. Springer, Berlin
13. Irani S, Gupta R, Shukla S (2002) Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In: Conference on design, automation and test in Europe, p 117
14. Khan SU (2009) A game theoretical energy efficient resource allocation technique for large distributed computing systems. In: International conference on parallel and distributed processing techniques and applications (PDPTA), pp 48–54
15. Khan SU (2009) A multi-objective programming approach for resource allocation in data centers. In: International conference on parallel and distributed processing techniques and applications (PDPTA), pp 152–158
16. 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 20(3):346–360
17. Kliazovich D, Bouvry P, Khan SU (2010) DENS: Data center energy-efficient network-aware scheduling. In: ACM/IEEE international conference on green computing and communications (GreenCom), pp 69–75
18. Laplante PA (2004) Real-time system design and analysis. Wiley, New York
19. Li L, Lai KK (2000) A fuzzy approach to the multiobjective transportation problem. Comput Oper Res 27(1):43–57
20. Liang T-F (2008) Fuzzy multi-objective production/distribution planning decisions with multi-product and multi-time period in a supply chain. Comput Ind Eng 55(3):676–694
21. Lorch JR, Smith AJ (2001) Improving dynamic voltage scaling algorithms with pace. In: 2001 ACM SIGMETRICS international conference on measurement and modeling of computer systems, pp 50–61
22. Luenberger D (1984) Linear and nonlinear programming. Addison-Wesley, Reading
23. Mejia-Alvarez P, Levner E, Mossé D (2004) Adaptive scheduling server for power-aware real-time tasks. ACM Trans Embed Comput Syst 3(2):284–306
24. Nathuji R, Isci C, Gorbatov E (2007) Exploiting platform heterogeneity for power efficient data centers. In: 4th international conference on autonomic computing, p 5
25. Pinel F, Pecero J, Bouvry P, Khan SU (2010) Memory-aware green scheduling on multi-core processors. In: 39th IEEE international conference on parallel processing (ICPP), pp 485–488
26. 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
27. Rusu C, Ferreira A, Scordino C, Watson A (2006) Energy-efficient real-time heterogeneous server clusters. In: 12th IEEE real-time and embedded technology and applications symposium, pp 418–428

28. Schrage L (1986) Linear, integer, and quadratic programming with LINDO. Scientific Press, South San Francisco
29. Stefanescu A, Stefanescu M (1984) The arbitrated solution for multi-objective convex programming. Rev Roum Math Pures Appl 29:593–598
30. Subrata R, Zomaya AY, Landfeldt B (2010) Cooperative power-aware scheduling in grid computing environments. J Parallel Distrib Comput 70(2):84–91
31. Wallenius J (1975) Comparative evaluation of some interactive approaches to multicriterion optimization. Manag Sci 21:1387–1396
32. Weiser M, Welch B, Demers A, Shenker S (1994) Scheduling for reduced cpu energy. In: 1st USENIX conference on operating systems design and implementation, p 2
33. Yu Y, Prasanna VK (2002) Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In: 9th international conference on parallel and distributed systems, p 341
34. Zangiabadi M, Maleki HR (2007) Fuzzy goal programming for multiobjective transportation problems. J Appl Math Comput 24(1):449–460