A Multi-objective Programming Approach for Resource Allocation in Data Centers

Samee Ullah Khan
Department of Electrical and Computer Engineering
North Dakota State University
Fargo, ND 58108
samee.khan@ndsu.edu

Abstract— *We model the process of a data center as a 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. A simple technique based on multi-objective goal programming is proposed that guarantees Pareto optimal solution with excellence in convergence process. The proposed technique also is compared with other traditional approach. The simulation results show that the proposed technique achieves superior performance compared to the min-min heuristics, and competitive performance relative to the optimal solution implemented in LINDO for small-scale problems.*

Keywords: Energy-efficient computing, distributed systems, multi-objective optimization.

1. Introduction

Data Centers are huge structures that house services for customers. Owing to their structural services they are continuously demanded for increase in throughput and reduced energy consumption. Energy-efficient techniques for managing a system at runtime can bring down the amount of energy it consumes. These management techniques are mostly for

- 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) [9], while the latter is called Dynamic Voltage Scaling (DVS) [12] (or Speed Scaling in the more theoretical literature [2]).

DPM considers a system (in the simplest case a processor) that can be in one of the two states, which 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 [4].

DVS on the other hand, seeks to exploit the convex relationship between the CPU supply voltage (that impacts the speed of execution) and the power consumption. The power consumption in CMOS circuits is given by $P = V^2 \times f \times C_{EFF}$, 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 QoS measures [22]. In this paper, we will investigate the joint optimization of energy consumption and response time. Because response time improves whence the 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.

The remainder of this paper is organized as following. The system model and problem formulation are discussed in Section 2. Section 3 provides some essential information pertaining to goal programming and details our proposed approach. Simulation results and related work are provided in Sections 4 and 5, respectively. Finally, in Section 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, $M = \{m_1, m_2, \cdots, m_m\}$. 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, which is approximately proportional to the frequency of the machine [14], [23].
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, $T = \{t_1, t_2, \cdots, t_n\}$. 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, which 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 $\frac{t_{ij}}{c_{ij}} = S_{ij}$. 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

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

$$\text{subject to } x_{ij} \in \{0, 1\}, \quad (2)$$

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

$$t_{ij} x_{ij} \leq d_i | x_{ij} = 1, \quad (4)$$

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

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

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 exists 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 above 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 [8]. Because, the multi-objective problem (described in this paper) has the objectives of optimizing instantaneous power and makespan that are are opposite to each other, we must choose to optimize both the objectives concurrently.

3. Goal Programming

3.1 A Generalized Goal Programming Procedure

Goal programming implicitly assumes that a desired goal is obtainable that can be used during each of the iteration of the solution convergence process through some high-level intervention [24]. 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 [10]. 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, which are analyzed to proceed towards 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 [11]. Below we describe a generalized procedure for goal programming.

Let the multi-objective problem to be solved be

$$\min \left(f_1(x), f_2(x), \cdots, f(_k(x)) \right), \quad (7)$$

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

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* [11]. 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 towards the desirable compromise solution. Moreover, they also determine the feasibility of the solution space.

Step 1: Set initial goals, $b = \{b_1, b_2, \cdots, k\}$. 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.

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

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

$$f_i(x) + w_i \times d_i^- - w_i \times d_i^+ = b_i, \tag{9}$$

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

$$d_i^- \leq 1,$$

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

where $w_i = b_i - f_{*i}$. The optimization of the MS would result in x^0 and f^0, which is known in the literature as the core solution [11]. 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 whence 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 2: 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* [24].

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 weightage for an 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 [6].

3.2 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).

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

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

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 [20]. 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 [13]. 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):

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

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

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

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 m-th 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.

3.3 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 must take in as an input the sets M and T with all machines initialized to their corresponding highest level of DVS, and produce a task to machine mapping.

To derive an upper and lower bound on the desired goal (corresponding to Step **0** of Section 3.1), 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 instantaneous power a mapped machine would consume, and vice versa.

The Kuhn-Tucker conditions derived in Section 3.2 set the initial goals corresponding to Step **1** of Section 3.1. They are not depicted in Algorithm 1 that describes the GP technique. Instead, implicitly, Steps **2** and **3** guide the solution towards a best possible compromise [21].

To develop a MS (corresponding to Step **2** of Section 3.1.), 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 [16]. 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 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 Section 3.1. 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 maped 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.

Input: T and M.
Output: Task to machine mapping.
Initialize: $\forall j, \mathrm{DVS}_j$ is set to the highest level.
while $T \neq \emptyset$ **do**
  $\mathcal{I} \leftarrow \mathrm{argmin}_i(d_i)$ **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
  $\mathcal{J} \leftarrow \mathrm{argmin}_j(\Delta)$;
  while $\{t_{\mathcal{I}\mathcal{J}} x_{\mathcal{I}\mathcal{J}} \leq d_{\mathcal{I}} | x_{\mathcal{I}\mathcal{J}} = 1\}$ **do**
    Reduce $\mathrm{DVS}_{\mathcal{J}}$ 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).

The GP heuristic as mandated in Section 3.1 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 the 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 Section 3.2). Moreover, PS ensure that optimum solution is identified (definition of PS, Section 3.1). 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 $\mathcal{O}\left(n^2 \log n + 3mn + mn \log m\right)$. Because it is assumed that $m \ll n$, the worst-case bound reduces to $\mathcal{O}\left(n^2 \log n\right)$.

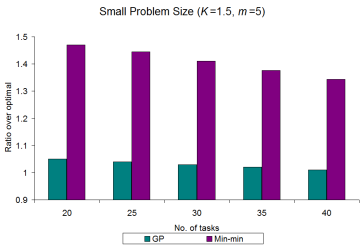
4. Simulations, Results, and Discussion

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 and the min-min heuristic [23]. (b) To measure the impact of system parameter variations. Based on the size of the problems, the experiments were divided in two parts.

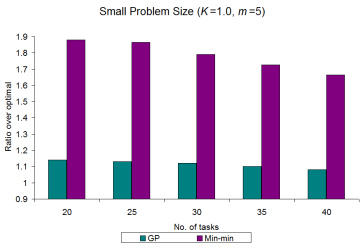
For small size problems, we used an Integer Linear Programming tool called LINDO [19]. LINDO is useful to obtain optimal solutions, provided the problem size is relatively small. Hence for small problem sizes, the performance of the proposed is compared against 1) the optimal solution using LINDO and 2) the min-min heuristic. The LINDO implementation and the min-min heuristic do not consider power as an optimization constraint; however, they are very effective for the optimization of the makespan. Thus, the comparison provides us with a wide range of results. On one extreme we have the optimal algorithm, on the other a technique which scales well with the corresponding increase in the problem size. For large size problems, it becomes impractical to compute the optimal solution by LINDO. Hence, we only consider comparisons against the min-min heuristic.

The system heterogeneity is captured by the distribution of the number of CPU cycles, c_{ij}, on different m_js. Let C denote the matrix composed by c_{ij}. The C matrix was generated using the coefficient of variation method described in [23]. The deadline, d_i, of task t_i was generated using the method described in [23]. For this study, we keep the architectural affinity requirements confined to memory. (Adding other requirements such as, I/O, processor type, etc. will bear no affect on our experimental setup or theoretical results.) Each machine is assigned a memory on random from within the range [500-5000] GB, while each task is associated a corresponding memory requirement on random from within the range [20-50] MB.

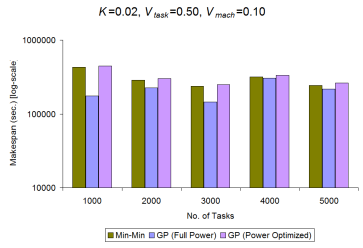
For small size problems, the number of machines was fixed at 5, while the number of tasks varied from 20 to 40. The number of DVS levels per machine was set to 4. The frequencies of the machines were randomly mapped from within the range [200MHz-2000MHz]. We assumed that the



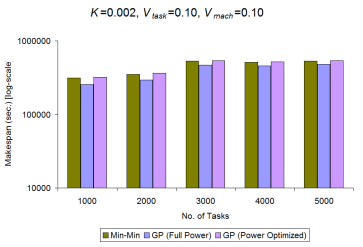
(a) Makespan ratio over the optimal.



(b) Makespan ratio over the optimal.



(c) Makespan.



(d) Makespan.

Fig. 1: Simulation results.

potential difference of 1mV across a CMOS circuit generates a frequency of 1MHz. For large size problems, the number of machines was fixed at 16, while the number of tasks varied from 1000 to 5000. The number of DVS levels per m_j was set to 8. Other parameters were the same as those for small size problems.

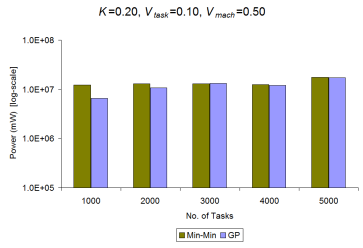
The experimental results for small size problems with K equal to 1.5 and 1.0 are reported in Figs. 1(a) and 1(b). These figures show the ratio of the makespan obtained from the two techniques and the optimal. The plots clearly show that the GP (proposed) technique performs extremely well and achieves a performance level of 10%–15% of the optimal when K was set at a very tight bound 1.0.

For large problem instances, first, we compare the makespan identified by the min-min and the GP technique. Since the min-min heuristic does not optimize power consumption, we compared the min-min with a version of GP that ran on full power and also compared it with the (original) version that optimized power. Figs. 1(c) and 3(d) show the relative performance of the techniques with various values of K, V_{task}, and V_{mach}. The results indicate that GP outperforms the min-min technique in identifying a smaller makespan when power is not considered as an optimization criteria. The performance of GP is notably superior to the min-min technique when the deadline constraints are relatively loose. It can also be observed that GP, when considering power as an optimization resource, identifies a task to machine mapping that produces a makespan that is within 5%-10% of the min-min technique. It was noticed that the relative performance of the min-min technique was much better for large size problems, compared with small size problems, because with the increase in the size of the C matrix, the probability of obtaining larger values of wis also increases. Moreover, the relative performance of GP was also much better for large size problems, compared with small size problems, because the DVS levels for the large problem size are twice more than the DVS levels for the small problem size.

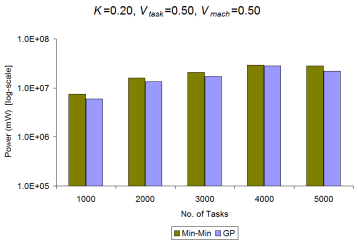
Next, we compare the power consumption of both the techniques. Figs. 2(a) and 2(b) reveal that on average the GP technique utilizes 60%–65% less power as compared to the min-min technique. That is a significant amount of savings considering that the makespan identified by GP is within 5%–10% of the makespan identified by the min-min technique.

5. Related Work

Most DPM techniques utilize instantaneous power management features supported by hardware. For example, Ref. [1] extends the operating system's power manager by an adaptive power manager (APM) that uses the processor's DVS capabilities to reduce or increase the CPU frequency, thereby minimizing the overall energy consumption [3]. The DVS technique at the processor-level together with a turn



(a) Power comsumption.



(b) Power consumption.

Fig. 2: Power consumption simulation results.

on/off technique at the cluster-level to achieve high-power savings while maintaining the response time is proposed in [18]. In [17], 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.

While the closest techniques to combining device power models to build a whole system has been presented in [5], our approach aims at building a general framework for autonomic power and performance management. Furthermore, while most power management techniques are either heuristic-based approaches [7], [15] or stochastic optimization techniques [22], we use goal programming to seek radically fast and efficient solutions compared to the traditional approaches.

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] T. F. Abdelzaher and C. Lu. Schedulability analysis and utilization bounds for highly scalable real-time services. In *7th Real-Time Technology and Applications Symposium*, p. 15, 2001.

[2] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pp. 520–529, 2004.

[3] R. Bianchini and R. Rajamony. Power and energy management for server systems. *IEEE Computer*, 37(11):68–74, 2004.

[4] D. P. Bunde. Power-aware scheduling for makespan and flow. In *8th ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 190–196, 2006.

[5] J. Chen, M. Dubois, and P. Stenström. Simwattch: Integrating complete-system and user-level performance and power simulators. *IEEE Micro*, 27(4):34–48, 2007.

[6] J. S. Dyer. Interactive goal programming. *Operations Research*, 19:62–70, 1972.

[7] T. Heath, B. Diniz, E. V. Carrera, W. M. Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 186–195, 2005.

[8] C. L. Hwang and A. S. M. Masud. *Multiple Objective Decision Making—Methods and Applications: A State-pf-the-Art Survey*. Springer Verlag, berlin, 1979.

[9] S. Irani, R. Gupta, and S. Shukla. 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, 2002.

[10] L. Li and K. K. Lai. A fuzzy approach to the multiobjective transportation problem. *Computers and Operations Research*, 27(1):43–57, 2000.

[11] T.-F. Liang. Fuzzy multi-objective production/distribution planning decisions with multi-product and multi-time period in a supply chain. *Computers in Industrial Engineering*, 55(3):676–694, 2008.

[12] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with pace. In *2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 50–61, 2001.

[13] D. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 1984.

[14] P. Mejia-Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *IEEE Transactions on Embedded Computing Systems*, 3(2):284–306, 2004.

[15] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *4th International Conference on Autonomic Computing*, p. 5, 2007.

[16] P. A. Laplante. *Real-Time System Design and Analysis*. John Wiley & Sons, 2004.

[17] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on Compilers and Operating Systems for Low Power*, 2001.

[18] C. Rusu, A. Ferreira, C. Scordino, and A. Watson. Energy-efficient real-time heterogeneous server clusters. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 418–428, 2006.

[19] L. Schrage. *Linear, Integer, and Quadratic Programming with LINDO*. Scientific Press, 1986.

[20] A. Stefanescu and M. Stefanescu. The arbitrated solution for multi-objective convex programming. *Revue Roumaine de Mathematical Pures et Appliquees*, 29:593–598, 1984.

[21] J. Wallenius. Comparative evaluation of some interactive approaches to multicriterion optimization. *Management Sciences*, 21:1387–1396, 1975.

[22] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *1st USENIX conference on Operating Systems Design and Implementation*, p. 2, 1994.

[23] Y. Yu and V. K. Prasanna. Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In *9th International Conference on Parallel and Distributed Systems*, p. 341, 2002.

[24] M. Zangiabadi and H. R. Maleki. Fuzzy goal programming for multiobjective transportation problems. *Journal of Applied Mathematical Computing*, 24(1):449–460, 2007.