Green Scheduling: Scheduling of Control Systems for Peak Power Reduction

Truong Nghiem, Madhur Behl, George J. Pappas and Rahul Mangharam
Department of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, USA
{nghiem, mbehl, pappasg, rahulm}@seas.upenn.edu

***Abstract*—Heating, cooling and air quality control systems within buildings and datacenters operate independently of each other and frequently result in temporally correlated energy demand surges. As peak power prices are 200-400 times that of the nominal rate, this uncoordinated activity is both expensive and operationally inefficient. While several approaches for load shifting and model predictive control have been proposed, we present an alternative approach to fine-grained coordination of energy demand by scheduling energy consuming control systems within a constrained peak power while ensuring custom climate environments are facilitated. Unlike traditional real-time scheduling theory, where the execution time and hence the schedule are a function of the system variables only, control system execution (i.e. when energy is supplied to the system) are a function of the environmental variables and the plant dynamics. To this effect, we propose a geometric interpretation of the system dynamics, where a scheduling policy is represented as a hybrid automaton and the scheduling problem is presented as designing a hybrid automaton. Tasks are constructed by extracting the temporal parameters of the system dynamics. We provide feasibility conditions and a lazy scheduling approach to reduce the peak power for a set of control systems. The proposed model is intuitive, scalable and effective for the large class of systems whose state-time profile can be linearly approximated.**

***Keywords*-Scheduling; Energy Systems; Peak Power Reduction; Load Balancing;**

I. Introduction

During a major sporting event such as the NFL Super Bowl which is watched by over 111 million people, the power supply companies are anxious. When a commercial break begins, most viewers open the fridge to get a drink and within a minute several million refrigerator compressors and microwaves trigger, causing massive spikes in energy demand. Human behavior is responsible for the high temporal correlation of energy demand, frequently causing peak power consumption. Similarly, at the scale of a single commercial building, heating, ventilating, air conditioning and refrigeration (HVAC&R) system, chiller systems, and lighting systems trigger concurrently resulting in power demand spikes several times in a day. Most commercial buildings are subject to peak power pricing which is 200-400 times that of the nominal power rate [1] and this uncoordinated behavior results in both expensive and inefficient power consumption.

While there exist several different approaches to load balance power consumption by load shifting and load shedding (e.g. [2]), they operate on coarse grained time scales and do not help in de-correlating energy sinks. The focus of this paper is on a new approach for fine-grained scheduling of

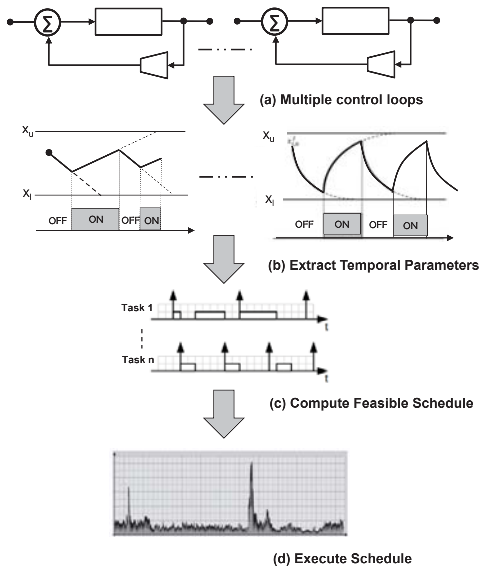

(a) Multiple control loops
x_u
x_l
OFF
ON
(b) Extract Temporal Parameters
Task 1
Task n
t
(c) Compute Feasible Schedule
(d) Execute Schedule

Fig. 1: From control to scheduling.

control systems within an aggregate peak power envelop while ensuring the custom climate conditions are maintained within the desired ranges. We achieve this by extracting the temporal parameters across multiple control loops and specifying their timing requirements within a global schedule with a peak power constraint(Fig. 1). While traditional real-time scheduling algorithms [3] may be applied to such resource sharing problems, they impose stringent constraints on the task model. Generally, real-time scheduling is restricted to tasks whose worst case execution times are largely a function of the system specification, are fixed, and are known a priori. While this simplifies the runtime complexity, for control systems it does not effectively capture the systems behavior whose operation is dependent on the plant dynamics, environmental conditions and initial system state.

For example, consider a commercial building, where the tasks are different control loops that maintain the temperature within a dead-band. The execution time associated with each control task is the duration for which energy is consumed by the task. Classical fixed priority and dynamic real-time scheduling are based on a fixed resource requirement for each task and are therefore able to arbitrate the use of the shared resource with explicit design-time admission control and feasibility requirements. When control processes are mapped to tasks, their execution time is a function of the plant state (e.g. dimensions of the room, ingress and egress airflow), environment conditions (e.g. outside weather, human occupancy), operational requirements (e.g. temperature/humidity set points, air quality) and the initial state of the system. These external factors contribute to *elastic* execution times where a system may have to perform more work, the longer the task's response time due to the natural degradation of the uncontrolled plant's state. This elastic task model is difficult to incorporate within the current formulation of real-time scheduling and thus prompts to provide a new perspective on the problem.

The primary contribution of this effort is in the formulation of a framework for scheduling energy consuming control systems that require a peak power constraint. For example, reducing the peak power consumption of a set of control tasks while ensuring that all tasks meet their stability and performance requirements. We approach this with a geometric interpretation of the system dynamics, where a scheduling policy is represented as a hybrid automaton and the scheduling problem is presented as designing a hybrid automaton. Tasks are constructed by extracting the temporal parameters of the system dynamics. We provide feasibility conditions and a lazy scheduling algorithm to reduce the peak power for a set of control systems. The proposed model is scalable and effective for the large class of systems whose state-time profile is linear.

In Section II, we formulate the task model that abstracts dynamic systems. Section III discusses the applicability of real-time scheduling approaches for scheduling of control tasks with elastic execution times. Section IV presents the geometric interpretation of tasks and the scheduling problem. We provide the schedulability analysis for the linear case in Section V, a lazy scheduling algorithm in Sections VI and VII which implements the proposed scheme. We conclude the paper with related work followed by a discussion in Sections IX and X along with a roadmap of our future work.

II. Task Model

In order to create a schedule for the dynamic systems that are being coordinated we first abstract them as tasks. At any time, each system must be in one of pre-defined operation modes, which governs its dynamics. Each system has a state variable (e.g., room temperature) that grows or decays with time according to the dynamics in its current mode. The state-time profile of a dynamic system determines how this state variable varies with time in each mode. Performance or safety specifications require that each system must operate within an acceptable region, defined by an upper threshold and a lower threshold for its state.

Each system can be abstracted as one task. A task T is a tuple (x, l, h, X_0, M, dyn) in which:

- $x \in \mathbb{R}$ is the continuous state variable;
- $l \in \mathbb{R}$ and $h \in \mathbb{R}$, where $l < h$, are the lower and upper thresholds of the state, respectively;
- $X_0 \subseteq [l, h]$ is the set of initial states;
- M is a finite set of operation modes;
- $dyn : M \times \mathbb{R} \to \mathbb{R}$ is a mapping that specifies the dynamics in each mode.

At any time $t \in \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ is the set of non-negative reals, the value of the task's state, denoted by $x(t)$, must always stay within $[l, h]$. The operation mode of T at time t is denoted by $m(t) \in M$. The task's dynamics satisfy the differential equation $\dot{x}(t) = dyn(m(t), x(t))$ with $x(0) \in X_0$.

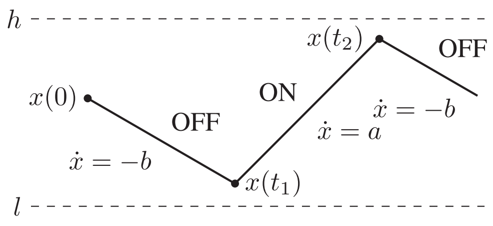


Fig. 2: State-time profile of a linear task.

In a discrete-time system, time t can only receive non-negative integer values and the mode of a task can only be changed at these discrete time instants. As a consequence, the task's dynamics is described by a difference equation instead of a differential equation: $x(t+1) = dyn(m(t), x(t))$.

In this paper, we consider tasks with two operation modes, $M = \{\text{ON}, \text{OFF}\}$. In addition, we only consider *linear tasks* which have linear state-time profiles. Specifically, the dynamics of a *linear task* is defined as:

$$\dot{x}(t) = dyn(m(t), x(t)) = \begin{cases} a & \text{if } m(t) = \text{ON} \\ -b & \text{if } m(t) = \text{OFF} \end{cases}$$

where a and b are strictly positive constants. When a task T is in ON mode, a is the rising slope (or growth rate) of its state. When T is in OFF mode, b is the falling slope (or decay rate) of its state. Fig. 2 illustrates the state-time profile of a linear task. Note that our definition of linear tasks is different from the notion of *linear systems* in control theory [4].

A task T is said to be *safe* if its state stays within the desired interval $[l, h]$ at all times; otherwise, T is said to be *unsafe*. Let $\mathcal{T}$ be a set of $n > 1$ tasks: $\mathcal{T} = \{T_i\}_{i=1,\ldots,n}$. A *scheduling policy* for $\mathcal{T}$ is an algorithm that specifies the mode of every task T_i in $\mathcal{T}$ at any time during the execution of the tasks. Besides ensuring that all tasks are safe, a scheduling policy usually has to satisfy other system-wide constraints such as a resource constraint which limits the number of tasks that can be executed at the same time. A task set $\mathcal{T}$ is *schedulable* by a policy π if π can schedule the tasks in $\mathcal{T}$ so that they are all safe and that all system-wide constraints are met. If $\mathcal{T}$ is schedulable by some scheduling policy, it is *feasible*; otherwise, i.e., $\mathcal{T}$ is not schedulable by any policy, it is *infeasible*. The problem of finding a scheduling policy for a feasible task set $\mathcal{T}$, often subject to additional constraints or optimality conditions, is a *scheduling problem*.

In the rest of this paper, we consider a set $\mathcal{T}$ of $n > 1$ linear tasks with two operation modes ($M = \{\text{ON}, \text{OFF}\}$). We also impose a system-wide resource constraint that at most one task can be in mode ON at any time.

III. Real-time Scheduling Interpretation

In this section, we compare the task model in Section II with that in real-time systems ([3]) and show that traditional fixed priority and dynamic scheduling policies, based on a fixed worst-case execution time, do not work well for this model.

In the traditional task model, a task is a sequence of jobs. Each job has a *release time*, an *execution time*, and a *deadline*. Our task model is based on the physical processes underlying the systems, thus it is more dynamic. A task is executed when it is switched ON by the scheduler. There is no notion of *release time* in our model because tasks are always available unless they are at their upper thresholds. When a task is OFF, its relative *deadline* is determined by system and environmental dynamics and its current state, and can vary from very short to very long. The *execution time* of a task is defined as the time between the moment it is switched ON and the moment it reaches its upper threshold if it remains ON. Clearly, the execution time of a task is not constant but dependent on its dynamics and its state. After a task is switched ON, its actual execution time can vary and is determined by the scheduler. This time will indirectly determine the deadline of the task after being switched OFF, since it determines the state of the task. In our task model, the temporal parameters of the tasks are not fixed but varying and highly dependent on the dynamics of the tasks. Furthermore, there are complex relations between the parameters.

Two well-known real-time scheduling algorithms for the traditional task model are the *rate-monotonic* (RM) and *earliest deadline first* (EDF). Because of the lack of *release time* and *task rate* in the proposed model, the RM algorithm does not translate to our task model. In addition, RM assumes a *fixed* worst case execution time for all tasks and is unable to factor in elastic execution times which are a function of environmental and process-related factors. On the other hand, since the notion of a deadline is well defined in our task model, the EDF algorithm can be used, in which the scheduler always switches ON the task with the earliest deadline. Nevertheless, it has different successes on different platforms.

Consider a continuous-time platform on which we schedule two tasks $\{T_1, T_2\}$ using the EDF algorithm. For the purpose of presentation, we scale these tasks so that $h_1 = h_2 = h$ and $l_1 = l_2 = l$. Let $d_i(t)$ denote the deadline of task T_i at time t. Without loss of generality, assume that initially the deadline of T_1 is earlier than that of T_2. Thus, initially, T_1 is ON and T_2 is OFF. As time t progresses, the deadline $d_1(t)$ of T_1 increases and approaches $d_2(t)$ which remains at $d_2(0)$. There are two possibilities that can happen:

1) State $x_1(t)$ reaches the upper threshold h before $d_1(t)$ reaches $d_2(t)$ (Fig. 3a): in this case, T_2 remains OFF while the EDF scheduler must switch T_1 OFF so as it does not exceed the upper threshold. However, after an infinitesimal time, T_1 becomes available again and must be switched ON. Then immediately it reaches h and must be switched OFF. This process is repeated infinitely fast in continuous time. Therefore, a Zeno phenomenon happens and the time cannot progress.
2) Deadline $d_1(t)$ reaches $d_2(t)$ before $x_1(t)$ reaches h (Fig. 3b): in this case, T_2 is switched ON and T_1 is switched OFF. Immediately $d_2(t)$ exceeds $d_1(t)$ and the scheduler must switch their modes, then the process repeats. Again, a Zeno phenomenon happens.

Therefore, the EDF algorithm does not work on an ideal continuous-time platform due to the Zeno phenomenon.

On a discrete-time platform where the tasks can only be switched at discrete time instants, the Zeno phenomenon can always be avoided. However, it is straightforward to see that the switching rates of the tasks can be high, in the worst case at every discrete instants. This is unfavorable in real systems, especially in HVAC or mechanical systems, because not only does it affect the system performance, it also reduces the lifetime of the equipment. Aperiodic server-based scheduling schemes such as the Constant Bandwidth Server (CBS) [3] which are designed to incorporate variable execution times, do not work well here as they build in their own hysteresis in servicing tasks and only provide a statistical guarantee for the overall system's stability.

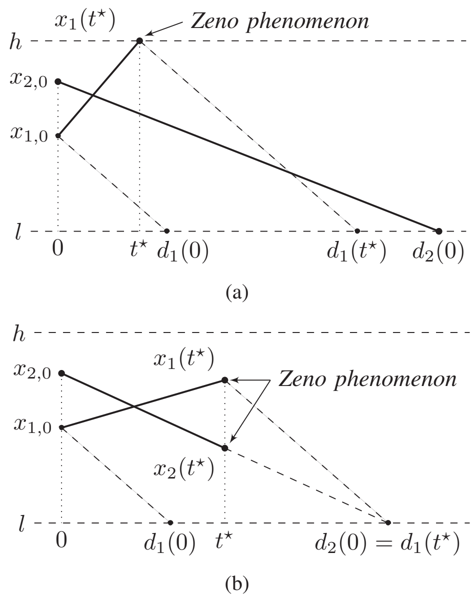


Fig. 3: Zeno phenomenon by EDF in continuous-time.

The failure and the unfavorable performance of traditional real-time scheduling algorithms for our task model motivate the development of a *lazy* scheduling algorithm, which is presented in the next section and also in Section VI.

IV. Geometric Interpretation

In this section, we present a geometric interpretation of tasks, scheduling policies and the scheduling problem, which provides a more intuitive and appropriate scheduling framework for control systems. For simplicity, we consider in this section two linear tasks T_1 and T_2 which are normalized so that their bounds are both $[0, 1]$.

Define a 2-dimensional state vector $x = [x_1, x_2]^T \in \mathbb{R}^2$. The state space of the system is the set $\mathbb{R}^2$. Any state $x(t)$ of the system at time t is a point in this state space. The invariant set of x is the unit square in $\mathbb{R}^2$ defined by the bounds of the two tasks. In Fig. 4, it is the grayed square. The set of initial states, not illustrated in Fig. 4, is a subset of the invariant set. There are three scheduling modes:

- **Mode 0**: T_1 and T_2 are OFF. The dynamics of the system is specified by vector v_0 in Fig. 4, that is if the system is in mode 0 for a duration Δ after time t then $x(t + \Delta) = x(t) + \Delta v_0$.
- **Mode 1**: T_1 is ON and T_2 is OFF. The dynamics of the system is specified by vector v_1 in Fig. 4.
- **Mode 2**: T_1 is OFF and T_2 is ON. The dynamics of the system is specified by vector v_2 in Fig. 4.

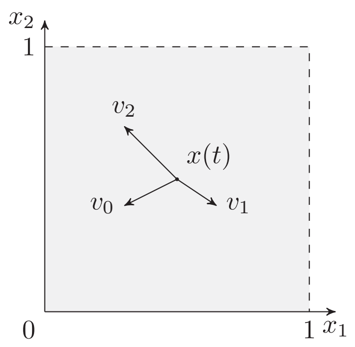


Fig. 4: State space of two tasks T_1 and T_2; the invariant set (gray square) and three mode vectors v_0, v_1, v_2.

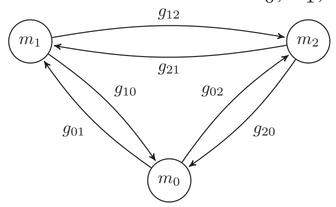


Fig. 5: Scheduling policy as hybrid automaton

The most important goal of a scheduling policy for these tasks is to keep $x(t)$ inside the invariant set using only the mode vectors v_0, v_1 and v_2. For a set of n linear tasks, the state space becomes n-dimensional and the mode vectors are v_i for $i = 0, 1, \dots, n$.

A. *Scheduling problem as hybrid automaton design*

Using the geometric interpretation, a scheduling policy can be represented as a hybrid automaton and the scheduling problem can be cast as designing a hybrid automaton [5]. The hybrid automaton for T_1 and T_2 has three discrete states corresponding to the three scheduling modes (Fig. 5). The directed edges between them represent the switchings between modes. A transition from one mode to another is caused by an event which occurs when the system state hits a guard associated with that edge. In Fig. 5, g_{ij} is the guard for the transition from mode i to mode j. Generally, a guard g_{ij} has the form $\hat{g}_{ij}(x) \le 0$ in which $\hat{g}_{ij} : \mathbb{R}^2 \to \mathbb{R}$ is a function mapping a state x to a real number. In many cases, a guard has a simple linear form $c^T x \le f$ for some vector c and some scalar f. A scheduling policy π for the task set is simply a set of guards $\{g_{ij}\}$, and the scheduling problem is equivalent to designing these guards. A discrete-time scheduling policy corresponds to a discrete-time version of the hybrid automaton defined above.

B. *Lazy scheduling policy*

A simple scheduling policy is the *lazy policy*: all tasks stay in their current modes as long as they are all safe, and only switch modes when their states hit a threshold. This policy corresponds to the following set of guards (for two tasks):

- g_{01} and g_{21} are both $(x_1 \le l_1)$;
- g_{02} and g_{12} are both $(x_2 \le l_2)$;
- g_{10} is $(x_1 \ge h_1 \wedge x_2 > l_2)$;
- g_{20} is $(x_2 \ge h_2 \wedge x_1 > l_1)$.

Fig. 6 illustrates the lazy policy for the two tasks in Fig. 4. The blue boundaries are the guards. The blue vectors represent

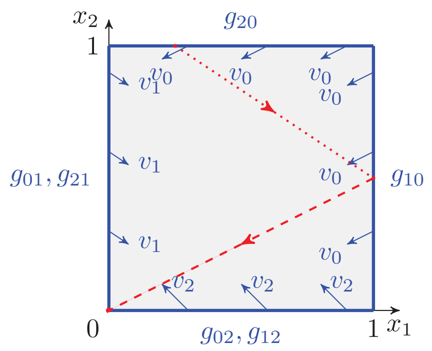


Fig. 6: Lazy scheduling policy for 2 tasks.

the dynamics of the system at the boundaries. The red points and red lines represent a critical situation when the system ends up at the origin, where both tasks must be switched ON immediately but only one of them can and the other will become unsafe. For these particular tasks, it is straightforward to see that the system is only in a critical situation if its state is on the red dashed line while it is in mode 0. Note that the red dotted line never happens because it requires mode 1 but it starts from a point on the g_{20} guard which will switch the system to mode 0.

V. Feasibility of Linear Tasks

In this section we look at theorems that govern the feasibility and infeasibility of a set of n linear tasks $\mathcal{T} = \{T_i\}$ as defined in Section II.

A. *Infeasibility result*

At time $t \ge 0$, let $t_{\text{on},i}$ be the total time since the beginning (i.e., $t = 0$) that task T_i is ON. Clearly, $0 \le t_{\text{on},i} \le t$ and the total time since the beginning that T_i is OFF is $t_{\text{off},i} = t - t_{\text{on},i}$. The resource constraint requires that

$$\textstyle\sum_{i=1}^n t_{\text{on},i} \le t. \tag{1}$$

The state of task T_i at time t can be computed as

$$x_i(t) = x_{0,i} - b_i t_{\text{off},i} + a_i t_{\text{on},i} = x_{0,i} - b_i t + (a_i + b_i) t_{\text{on},i}$$

where $x_{0,i} = x_i(0)$. Define new variables $\hat{x}_i = \frac{x_i - x_{0,i}}{a_i + b_i}$, $i = 1, \dots, n$. We have that

$$\hat{x}_i(t) = -\frac{b_i}{a_i + b_i} t + t_{\text{on},i} = -d_i t + t_{\text{on},i}$$

in which $d_i = \frac{b_i}{a_i + b_i} > 0$. The safety constraint requires that $\hat{l}_i \le \hat{x}_i \le \hat{h}_i$ where $\hat{l}_i = \frac{l_i - x_{0,i}}{a_i + b_i}$ and $\hat{h}_i = \frac{h_i - x_{0,i}}{a_i + b_i}$.

Taking the sum of all $\hat{x}_i(t)$ gives

$$\hat{x}(t) = \textstyle\sum_{i=1}^n \hat{x}_i(t) = -\sum_{i=1}^n d_i t + \sum_{i=1}^n t_{\text{on},i} \tag{2}$$

with the constraint that $\hat{l} = \sum_{i=1}^n \hat{l}_i \le \hat{x}(t) \le \hat{h} = \sum_{i=1}^n \hat{h}_i$. Let $d = \sum_{i=1}^n d_i > 0$. Using (1) and (2), we obtain the following inequality

$$\hat{x}(t) \le (1 - \textstyle\sum_{i=1}^n d_i)\, t = (1 - d) t. \tag{3}$$

Inequality (3) gives us a condition on the infeasibility of the set of tasks, as stated in the following theorem.

Theorem 1. *If $d > 1$, the set $\mathcal{T}$ of linear tasks is infeasible.*

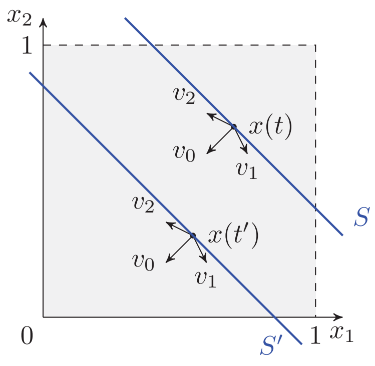


Fig. 7: Geometric intuition of Theorem 1

Proof: If $d > 1$ then $1 - d < 0$. By inequality (3), $\hat{x}(t)$ is bounded above by a strictly decreasing function $(1-d)t$. By time $t = \hat{l}/(1-d) \geq 0$ at the latest, $\hat{x}(t)$ will violate the safety constraint, regardless of the scheduling policy. ■

The geometric interpretation of tasks (section IV) provides a good intuition of Theorem 1. If there exists a surface S containing $x(t)$ such that all mode vectors at $x(t)$ are on the same side of S then clearly for any scheduling policy, state x will go further to this side and will eventually go out of bound. This is illustrated in Fig. 7 where the state is moved from $x(t)$ to $x(t')$ at some later time $t' > t$ and the surface S is shifted to S' towards the lower left corner of the invariant set. The condition $d > 1$ in Theorem 1 implies this situation as shown in the following theorem.

Theorem 2. *If $d > 1$, for any state x, there exists a surface S containing x such that all mode vectors v_i, $i = 0, 1, \ldots, n$, are on the same side of S.*

The proof for this theorem is straightforward by choosing space S with normal vector v:

$$v = -\begin{bmatrix} \frac{1}{a_1+b_1} & \frac{1}{a_2+b_2} & \cdots & \frac{1}{a_n+b_n} \end{bmatrix}^T$$

then showing that for all mode vectors v_i, $i = 0, 1, \ldots, n$, we have $\langle v_i, v \rangle > 0$ where $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors.

B. Feasibility result

Knowing that $d > 1$ implies infeasibility of the task set (Theorem 1), it is natural to ask whether $d \leq 1$ implies feasibility. In this section, we provide an answer to that question by presenting a feasibility result for a set of linear tasks on a continuous-time platform where there is no technical constraint on when a task can be scheduled. Hence, on this platform, a task can be switched ON and OFF arbitrarily fast or slow as long as it satisfies the safety constraint.

We proceed with the main feasibility theorem.

Theorem 3. *If d_i is rational for every $i = 1, \ldots, n$ and $d = \sum_{i=1}^n d_i \leq 1$ and at most one task starts at its lower threshold then the task set is feasible on a continuous-time platform.*

Proof: If more than one task start at their lower thresholds then the task set is infeasible because at least one task will violate its safety constraint regardless of the schedule. We can assume that at the beginning, no tasks start at their thresholds, i.e., $l_i < x_i(0) < h_i$ for $i = 1, \ldots, n$. Indeed, if there are tasks starting at their thresholds, among which at most one can start at its lower threshold, we can always schedule the tasks for a short period of time $\tau > 0$ so that $l_i < x_i(\tau) < h_i$ for all $i = 1, \ldots, n$.

To prove feasibility, we will construct a periodic schedule for the tasks, specified by a time period $\Delta_T^\star$ and an ordering sequence $\rho = (\rho_1, \ldots, \rho_Q)^\omega$ of length Q, where each $\rho_k \in \{0, 1, \ldots, n\}$ indicates which task is ON during interval k, and the superscript ω means that the sequence is repeated indefinitely. If $\rho_k = 0$ then all tasks will be OFF. Define time instants $t_k = k\Delta_T^\star$, $k \geq 0$. The sequence ρ is designed so that state x always repeats after one recurrence of ρ, i.e., $x(t_{k+Q}) = x(t_k)$ for any $k \geq 0$. The time period $\Delta_T^\star$ is chosen so that starting from a safe initial state $x(0)$, $x(t_k)$ is safe for every $k = 1, 2, \ldots, Q-1$. Therefore, for any $k \geq 0$, $x(t_k) = x(t_{(k \bmod Q)})$ is safe. Thus, the task set is (safely) schedulable by the constructed periodic schedule.

a) Constructing sequence ρ: For each task T_i, since d_i is rational, we can write $d_i = \frac{p_i}{q_i}$ where $p_i, q_i \in \mathbb{N}$. Let Q be a common multiple of all q_i, for $i = 1, \ldots, n$, then we can write $d_i = \frac{P_i}{Q}$ for some $P_i \in \mathbb{N}$. We assign to each ρ_k in ρ an integer between 0 and n so that for each integer $i \in \{1, \ldots, n\}$, the number of times it appears in ρ is exactly P_i. Because $d = \sum_{i=1}^n d_i = \sum_{i=1}^n \frac{P_i}{Q} \leq 1$, we have $\sum_{i=1}^n P_i \leq Q$. It follows that we can always construct ρ to satisfy the above condition. To prove the periodicity of x, we only need to show that $x(t_Q) = x(0)$ because it implies $x(t_{Q+1}) = x(t_1)$, $x(t_{Q+2}) = x(t_2)$, and so on. Indeed, for each task T_i, we have

$$\begin{aligned} x_i(t_Q) &= x_i(0) + P_i\Delta_T^\star a_i - (Q - P_i)\Delta_T^\star b_i \\ &= x_i(0) + Q\Delta_T^\star\left(d_i a_i - (1-d_i)b_i\right) \end{aligned}$$

in which $d_i a_i - (1-d_i)b_i = 0$. Thus $x(t_Q) = x(0)$.

b) Choosing $\Delta_T^\star$: For each step k, $k = 1, 2, \ldots, Q-1$, we will find the maximal time period $\Delta_{T,k} > 0$ so that $x(t_k)$ is safe. Then we can simply choose $\Delta_T^\star = \min\{\Delta_{T,1}, \ldots, \Delta_{T,Q-1}\} > 0$. Let $\beta_{i,k}$ be the number of times T_i is ON from the beginning to step k. In particular, it is defined recursively as follows: $\beta_{i,0} = 0$, $\beta_{i,k+1} = \beta_{i,k} + 1$ if $\rho_{k+1} = i$ and $\beta_{i,k+1} = \beta_{i,k}$ if $\rho_{k+1} \neq i$. Then we have

$$\begin{aligned} x_i(t_k) &= x_i(0) + \beta_{i,k}\Delta_T^\star a_i - (k - \beta_{i,k})\Delta_T^\star b_i \\ &= x_i(0) + \Delta_T^\star\left[\beta_{i,k}\left(a_i + b_i\right) - kb_i\right]. \end{aligned}$$

Let $\gamma_{i,k} = \beta_{i,k}\left(a_i + b_i\right) - kb_i$. We can compute the maximal time period for $x_i(t_k)$ to be safe as

$$\Delta_{T,k}^i = \begin{cases} (h_i - x_i(0))/\gamma_{i,k} & \text{if } \gamma_{i,k} > 0 \\ (l_i - x_i(0))/\gamma_{i,k} & \text{if } \gamma_{i,k} < 0 \\ +\infty & \text{if } \gamma_{i,k} = 0 \end{cases}$$

Because $l_i < x_i(0) < h_i$, $\Delta_{T,k}^i > 0$. Then for all $0 < \Delta_T^\star \leq \Delta_{T,k}^i$, $l_i \leq x_i(t_k) \leq h_i$. With $\Delta_{T,k} = \min_i \Delta_{T,k}^i > 0$, it is guaranteed that $x(t_k)$ is safe for all $0 < \Delta_T^\star \leq \Delta_{T,k}$. Therefore we choose $\Delta_T^\star = \min_{k=1,\ldots,Q-1}\Delta_{T,k} > 0$. ■

VI. The Lazy Scheduling Algorithm

We now describe the lazy scheduling algorithm which ensures that a set of tasks always remain safe given that they satisfy certain initial conditions. The algorithm is termed as lazy, because switching decisions are made only if a task

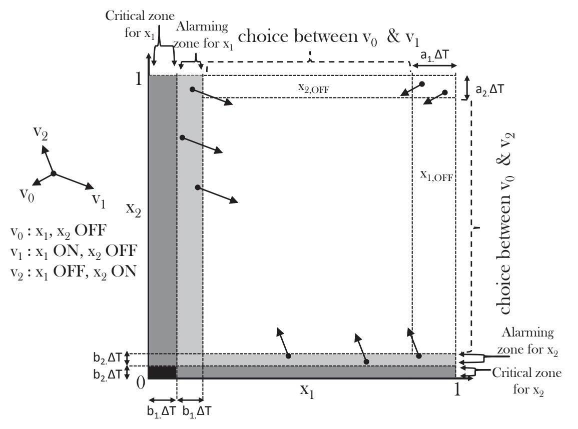


Fig. 8: Lazy Scheduling Algorithm for the 2D case

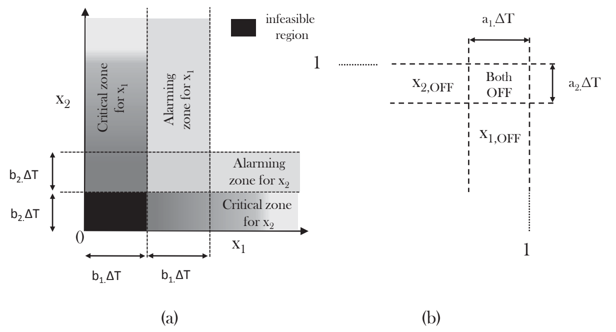


Fig. 9: Zoomed in view of regions of intersection of zones.

is approaching its thresholds. Time is discretized for the remaining paper since for practical systems time is discrete. We assume that for each task the growth rate is strictly greater than the decay rate (i.e. $a_i > b_i$). This is a reasonable assumption since zones within buildings tend to heat faster than they cool. Consider the normalized state space of two tasks as shown in Figure 8. For each task T_i, the following three zones are defined:

1) **Critical zone**: The Critical zone of a task T_i is the zone in which the task must remain ON for the next time step or else the task will violate its lower threshold in the next iteration. The width of the Critical zone of task T_i is given by $b_i\Delta_T$. A task is said to be *Critical* if its state is within its Critical zone.
2) **Alarming zone**: The Alarming zone of a task T_i is the zone in which keeping T_i OFF for the next iteration will take it into its Critical zone. The width of the Alarming zone of T_i is $b_i\Delta_T$, same as the Critical zone. A task is said to be *Alarming* if its state is within its Alarming zone.
3) **OFF zone**: For every task T_i, the OFF zone is the zone in which the task must remain OFF for the next iteration or else it will violate its upper threshold in the next iteration. The width of the OFF zone of task T_i is $a_i\Delta_T$.

As can be seen in Figure 8, there are regions of intersections between different zones of the tasks. A zoomed in view of these regions is shown in Figures 9: (a) & (b). The lower left corner of Figure 9(a) shows the region where the Critical zones for the two tasks intersect. This is termed as the *infeasible* region because if the system enters this region, then by definition, both the tasks will have to be switched ON for the next time step. Under the constraint that at most one task can remain ON, this would imply that one of the tasks will violate its lower threshold in the next time step.

Proposition 4. *Under the assumption that at most one task can remain ON, the two task system is not schedulable by any policy if the the initial state of the system is within the infeasible region, as defined above.*

The next region of interest is the region of intersection of the Alarming zones of the tasks. If the state of the system lies in this region at any point in time, then by definition of an Alarming zone, one of the tasks will become Critical in the next time iteration as only one task will be switched ON. The decision as to which tasks remains ON, can be random or based on optimizing a cost function. The third region of intersection is between the OFF zones for the tasks (Figure 9(b)). If the system enters this region then both the tasks must be switched OFF for the next iteration. There are some other regions of intersection as well, namely the ones between OFF zones and Critical zones. But in the algorithm, priority of a task being Critical or Alarming is always higher then the OFF state, so these regions are not of importance.
Given that the initial state of the system does not lie in the infeasible region, algorithm 1 can be used to maintain feasibility of the tasks while optimizing a cost function defined over the set of tasks. The algorithm works by using the

Algorithm 1 Lazy Scheduling Algorithm for n tasks

```
1: loop
2:   $S \leftarrow \phi$ {initialize switch queue as empty}
3:   read the state $x(t) = [x_1(t)...x_n(t)]$ of the system
4:   for each $T_i$ do
5:     if $T_i$ is in its OFF zone then
6:       switch it OFF in the next iteration
7:     else if $T_i$ is Alarming or Critical then
8:       $S \leftarrow S \cup \{T_i\}$ {add $T_i$ to $S$}
9:     end if
10:  end for
11:  if $|S| > 0$ then
12:    if any task in $S$ is Critical then
13:      switch it ON in next iteration
14:    else
15:      pick a task from S (randomly or based on optimization of some cost function) to switch ON
16:      switch OFF previously ON task
17:    end if
18:  end if
19: end loop
```

Alarming zones of the tasks as barriers that prevent the tasks from entering their Critical zones at any point in time. The working of the algorithm for a two task system is shown in Figure 8. Whenever the system enters the Alarming zone of a task, the system is switched into a mode (v_0, v_1 or v_2) such that the Alarming task will exit its Alarming zone in the next time iteration. This is becasue of the assumption that $a_i > b_i$. Within the OFF zones of the tasks, there is an option to either switch everything OFF or keep some tasks ON (which are not in their OFF zone). In our algorithm all tasks are kept OFF whenever any task is in its OFF zone provided no other task is Alarming or Critical during that time step. For scheduling n tasks, under the constraint that at most one task can remain ON at any time, $n-1$ Alarming zones are required. The Lazy Scheduling Algorithm is safe by construction. In the next section, the lazy scheduling algorithm is implemented for a set of tasks and the reduction in peak power consumption is simulated.

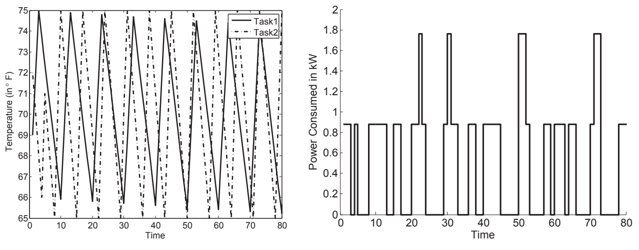
VII. Simulation and Results

In this section it is shown how the lazy scheduling algorithm can reduce the peak power consumption of a building by decorrelating different systems such that no two systems consume power at the same time. To setup the simulation the following models are required:

1) A Task Model
2) Power Consumption Model
3) Electricity Pricing Model

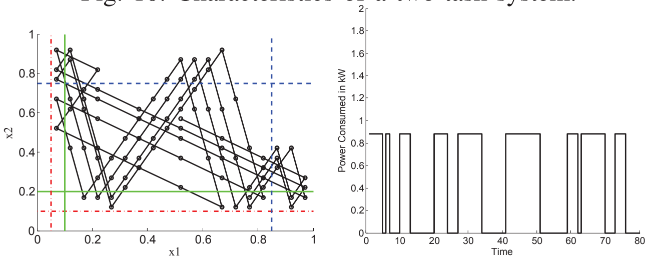
The task model is the same as described earlier in Section II. The tasks here represent heating and cooling systems. We approximate the temperature dynamics of such systems by linear tasks. Each system switches ON or OFF to maintain the temperature inside a zone within the upper and lower temperature thresholds. The state of the task in this case is the temperature of the zone. Multiple tasks means multiple such systems, each controlling a different zone.

The power consumption model simulates how much power is consumed by the heating and cooling processes. It is assumed that the heating system operates with a power of 12000 BTU/h or 3.517 kW. For simplicity, it is also assumed that cooling occurs through heat loss and does not consume any extra power, therefore turning the heating system OFF is equivalent to cooling the system. In an actual building, the cooling system would also consume some energy. In our model that would just mean assigning a power consumption value to the OFF state of the task as well. Further, without loss of generality, it is assumed that each zone operates within the temperature thresholds of $65°F - 75°F$. The time scale of tasks is specific to the dynamics of a system, but for the purpose of this paper it is assumed that each discrete time step of the algorithm (in Algorithm 1) is of 15 minute duration. Figure 10a shows the behavior of a two task system when each system is allowed to operate independently of the other. Each system maintains the temperature of a zone within the temperature range centered around $70°F$. When both the systems are ON simultaneously, their power consumption aggregates, resulting in multiple peaks in the power consumption of the system. This can be can be seen in Figure 10b. The value of the peak power consumed is 1.758 kW, averaged over multiple runs of the system starting from different initial states. The total energy consumed on average is 50.11 kWh. When we run the same set of tasks with the lazy scheduling algorithm, the average value of the peak power consumption reduces to 0.879 kW (Figure 11b). The total energy consumed in this case is 45.72 kWh on average. While there is a marginal decrease in the total energy consumption, the decrease in peak power demand is significant. This behavior is expected since



(a) State-Time behavior (b) Power consumption

Fig. 10: Characteristics of a two-task system.



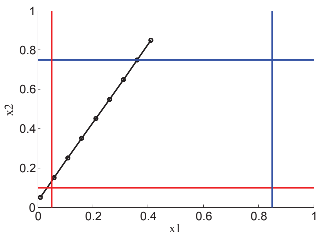
(a) State-space trajectory. (b) Power consumption.

Fig. 11: Two-task system with the lazy scheduler.

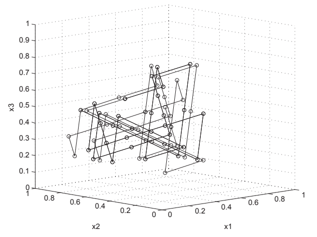
the lazy scheduling algorithm reduces peak power demand by ensuring that only one system consumes power at any time. Figure 11a shows the inner workings of the lazy scheduler in the normalized state space for the two-task system. As described in Sections V and VI, the objective of the algorithm is to keep the tasks bounded within the operational thresholds. This can be seen in the figure, as the system always operates within bounds.The growth and decay slopes of the tasks were set such that they adhere to Theorem 1.

As can be seen from Figures 10b and 11b, there is a 50% reduction in the peak power consumption. This is a huge improvement, especially because electricity utility companies charge commercial customers based on the peak power consumption. At this point, it is important to discuss how demand based pricing works in practice. If the peak power demand is denoted by Pk and the total energy consumption is denoted by Tot, the electricity bill is computed as $Bill = (p_u \times Tot) + (p_d \times Pk)$. Here p_d is the demand price which is much higher than p_u, the usage price (often more than 100 times higher). To give an example, electric utility companies in Pennsylvania, USA charge approximately 240 times higher prices for the peak demand than for the energy usage ([1], [6]). The high penalty for peaks in power consumption combined with high prices means that reducing the peak power not only saves electricity but also makes a system highly cost effective.

In Section VI, the importance of adding Alarming zones in the system, as barriers to prevent a task from entering its Critical zone, was emphasized. This can be verified by simulation for a two task system without the presence of an Alarming zone. In the absence of an Alarming zone there always exist a set of states in the system that can drive the system towards infeasibility (when both tasks become Critical at the same time). One such case is shown in Figure 12a where the system enters the infeasible region because no Alarming zone is present. We verify through simulations that the algorithm works for higher number of tasks. Figure 12b shows the working of the algorithm in the normalized state



(a) No Alarming zone



(b) State-Space Trajectory 3D case

Fig. 12

space for three tasks. It can be seen that the tasks always remain bounded. As the number of tasks increases the value of the peak power consumed also increases as multiple tasks consume power at the same time. Therefore, when the lazy scheduling algorithm decorrelates the set of tasks to run one at a time, the corresponding reduction in peak is also higher.

VIII. Related work

Over the past few decades, majority of the work done on real-time scheduling algorithms for single and multiprocessor systems builds upon the seminal work of Liu and Layland in 1973 [7]. Large classes of fixed priority and dynamic deadline-based scheduling have been proposed for CPU-centric task scheduling. A lot of work has also been done towards energy efficient CPU scheduling using Dynamic Voltage Scaling (DVS) and energy aware task allocation ([8],[9] and [10]). The integration of control and scheduling has been investigated by [11],[12] and [13], where extensions of real-time scheduling approaches have attempted to capture control task dynamics. Our approach is specifically focused on energy consuming control systems with a system-wide resource constraint and departs from a CPU-centric view to a PSU-centric (Power Supply Unit) resource allocation. The extension of traditional real time scheduling algorithms like EDF and RMS, for activation of electrical loads is used in [14] and [15] by assuming a periodic task activation model for the physical system. Although the periodic task model allows for the use of traditional scheduling algorithms but the underlying assumption that electrical loads need to switch periodically (and not based on state feedback) makes the system overly constrained and less flexible to changes in system dynamics. Model predictive control (MPC) has been used for optimal control of energy-efficient buildings [16], data centers [17], and other energy systems. Our approach focuses on a higher level task abstraction and though not as optimal as MPC, our scheduling algorithms are simpler to implement and do not require high computational power – one disadvantage of MPC.

IX. Limitations

This work is the initial step towards developing scheduling algorithms for peak power reduction of energy consuming systems such as buildings, datacenters etc. We have presented analysis for systems with dynamics which can be linearly approximated but we aim to extend the task model to incorporate more realistic system dynamics. Extensions to this work will also include incorporating dynamic pricing models, operational efficiencies and task-specific cost functions for system-wide optimization to the task model and implementing comprehensive scheduling policies. We are also investigating the case of scheduling with multiple tasks (k out of n) ON at the same time when the feasibility condition is not met.

X. Conclusion

A geometric formulation for the scheduling of multiple control systems is presented. Through feasibility and schedulability analysis, the proposed approach is shown to be effective in reducing aggregate resource usage by coordinating demands across tasks while ensuring the correct operation of all tasks. The proposed work is an initial step in the direction of resource and energy efficient coordination of control systems. Although the results in this paper are presented for systems resembling heated zones or buildings, they can be easily extended to cooling systems (e.g. in datacenters) and more generally to any set of energy consuming controls systems that require a peak power constraint.

References

[1] TRF Energy, "Understanding PECO's general service tariff."

[2] A. Cohen and C. Wang, "An optimization method for load management scheduling," *Power Systems, IEEE Transactions on*, vol. 3, no. 2, pp. 612 –618, May 1988.

[3] J. W. S. Liu, *Real-time Systems*. Prentice Hall, 2000.

[4] W. J. Rugh, *Linear System Theory*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[5] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1993, vol. 736, pp. 209–229.

[6] PECO, "How to understand your bill," February 2011.

[7] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, pp. 46–61, January 1973.

[8] T. AlEnawy and H. Aydin, "Energy-aware task allocation for rate monotonic scheduling," in *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, 2005, pp. 213 – 223.

[9] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *Computers, IEEE Transactions on*, vol. 53, no. 5, pp. 584 – 600, May 2004.

[10] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, ser. SOSP '01. New York, NY, USA: ACM, 2001, pp. 89–102. [Online]. Available: http://doi.acm.org/10.1145/502034.502044

[11] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "Trade-off analysis of real-time control performance and schedulability*," *Real-Time Systems*, vol. 21, pp. 199–217, November 2001.

[12] A. Cervin and J. Eker, "The control server: A computational model for real-time control tasks," in *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, July 2003, pp. 113–120.

[13] R. Chandra, X. Liu, and L. Sha, "On the scheduling of flexible and reliable real-time control systems," *Real-Time Systems*, vol. 24, pp. 153–169, March 2003.

[14] E. B. Tullio Facchinetti and M. Bertogna, "Reducing the peak power through real-time scheduling techniques in cyber-physical energy systems," in *Proceedings of the First International Workshop on Energy Aware Design and Analysis of Cyber Physical Systems (WEA-CPS)*, 2010.

[15] M. R. Marco Della Vedova and T. Facchinetti, "On real-time physical systems," in *Proceedings of the 18th International Conference on Real-Time and Network Systems (RTNS)*, 2010, pp. 41–49.

[16] F. Oldewurtel, A. Parisio, C. N. Jones, M. Morari, D. Gyalistras, M. Gwerder, V. Stauch, B. Lehmann, and K. Wirth, "Energy efficient building climate control using stochastic model predictive control and weather predictions," in *American Control Conference*, Jun. 2010, pp. 5100 –5105.

[17] L. Parolini, E. Garone, B. Sinopoli, and B. H. Krogh, "A hierarchical approach to energy management in data centers," in *IEEE Conference on Decision and Control*, December 2010.