Power-Managed Data Centers for Sustainable Computing

Emi Zeger, Nicholas Bambos, Mert Pilanci
Department of Electrical Engineering, Stanford University

***Abstract*—Data centers increasingly consume large amounts of power. In this paper, we develop an efficient and scalable power management scheme, based on optimizing the trade-off between a processor's speed and its response/delay time to process jobs. Processors/servers are lightly coordinated by a system manager that sets an internal power "price" signal to discourage excessive power usage. The stability of the power management scheme depends on the power pricing function, and a rule of thumb is given for pricing functions that guarantee stability. The analysis is consistent with simulations, which demonstrate the behavior of the scheme. It can be implemented in a decentralized manner across servers/processors, making it scalable to large data centers.**

I. INTRODUCTION

Demand for large-scale computing (e.g. for machine learning, model training, AI, the Internet-of-Things) is increasingly pushing (cloud-) computation into data centers. This has led to those growing in massive proportions, with 'hyperscale' data centers (containing hundreds of thousands of servers) being built in recent years [1]. As a result, data centers are increasingly becoming severely power-constrained [2] [3], already consuming about 1% of energy globally [4]. Much of the energy consumption is driven by CPUs and the cooling systems required to prevent them from overheating [3]. Technological progress in hardware has made data centers more energy-efficient, but new advancements are in demand to continue making data centers sustainable, as computing scales up [4]. In this paper, we develop a power management algorithm that 1) accommodates the processor/server demand for power, while discouraging its overuse, 2) is stable (power levels converge and don't oscillate or explode over time) and 3) is implementable at scale.

Various methods have emerged for managing power consumption in computing, with one promising approach being Dynamic Voltage and/or Frequency Scaling (DVFS) [5]. Chips can save energy [6] by reducing voltage and/or decreasing clock frequency [7]. Power consumption increases super-linearly with voltage and therefore it is more power-efficient to slow down CPU or chip clock frequency; however, this causes computation jobs or tasks to take longer to process. This creates a trade-off between power usage and computation job processing speed (or more generally, performance), as identified in [8]. To optimize this trade-off [9] proposes minimizing the joint cost of the task completion time plus energy usage. Motivated by technologies developed to use DVFS to prolong battery life for Windows XP operating systems, [10] develops algorithms to implement this in an efficient manner online, as jobs arrive with known completion times (under the fastest CPU rate). In contrast, [11] studies algorithms that scale the CPU speed based on just the number of current job arrivals without knowing their job sizes. Instead of considering each individual job's arrival and time left to finish, our paper focuses on the *average* completion time of jobs as a function of power.
DVFS has been found to save about 23% of energy in data centers in [12], which simulated applying DVFS when the CPU was being underutilized. There were especially high power savings for Webmail tasks [12]. In addition to DVFS, there are other related approaches for energy reduction in data centers, including switching servers on and off [13] or allocating servers to customers in a bidding system [14] to match demand and maximize the data center's profit, while minimizing energy consumption.

Power control algorithms to manage similar trade-offs have been developed in wireless communication [15] and smart grids [16], where real-time, dynamic pricing of power has been explored to regulate energy consumption. These approaches were combined in [17], which developed a distributed power control architecture for smart grids. In [17], electric grid operators respond to increased power demand by setting higher prices. At every time period, each user (such as a home) buys the optimal amount of power from the grid to balance its own trade-off between delay and electricity cost. Then, the grid operator responds by updating the power price.

Inspired by [17], we develop a processor and server power management scheme for data centers that can accelerate the processor/server speed using DVFS. We provide a general, scalable algorithm that manages the power versus latency trade-off. Both provable results and simulations are given to analyze stability, and a rule of thumb is provided for the algorithm parameters to ensure that the system converges.

The paper is organized as follows. Section II first describes the system model, and then defines the costs for latency (Section II-A) and power (Section II-B). Using these costs, Section II-C describes a power control algorithm and the system evolution under it. Section III and Section IV prove the main result that the system converges. Section V describes simulations and numerical performance analysis.

Notation: Given a set $\mathcal{C} = \{1, 2, \cdots, C\}$, and scalars x_c indexed by $c \in \mathcal{C}$, we stack them into a vector denoted as $(x_c, c \in \mathcal{C}) := (x_1, \cdots, x_C) \in \mathbb{R}_+^C$. For a function f, its inverse is denoted as $[f]^{-1}$. Define $[x]^+ = x$ for $x \geq 0$ and 0 otherwise. The composition of f and g, and g with itself n times, are denoted as $f \circ g$ and g^n, respectively.

II. Distributing Power to Computation Queues

Jobs sent to a data center come in different classes, differentiated by characteristics such as arrival rate and job priority. Consider a model in which a group of servers (perhaps virtual) is assigned to each class, and for simplicity let us abstract these groups to one "net" server per class. We consider C computation job classes, indexed by $c \in \mathcal{C} = \{1, 2, ..., C\}$. Arriving computations jobs of each class c are modeled as arriving at random times and being stored in a buffer or queue, awaiting execution; that is, there is one queue per class, also indexed by $c \in \mathcal{C}$.

We define the *response time* R_c to be the average sojourn time (from arrival to departure) of jobs through queue c. In traditional systems, CPUs are clocked at a fixed rate so that the response time cannot be altered by changing power consumption. However, our power management algorithm proposes to use DVFS, which makes the response time a function $R_c(p)$ of the power $p \geq 0$ consumed by class c. For each class c, there is an $\epsilon_c > 0$ such that $R_c(p) = \infty$ for all $p \leq \epsilon_c$, because, 1) when the server processing rate falls below the workload arrival rate due to low power usage by a queue, then the sojourn time in that queue will explode to infinity in that queue over time (see $M/M/1$ queue discussion below), and 2) even when the server has zero processing rate, it could be in stand-by or (light) sleep modes where it is still burning some (low) power. Let $P_\epsilon = \sum_{c \in \mathcal{C}} \epsilon_c$ be the minimum power used by all servers. Assume that for each class $c \in \mathcal{C}$ and $p \in (\epsilon_c, \infty)$, $R_c(p)$ is positive, strictly decreasing, strictly convex, and twice differentiable. We make $R_c(p)$ decreasing since processing gets faster and wait times decrease when servers run on more power.

For concreteness and intuition, consider the simplest scenario, a $M/M/1$ queue. We will return to this example throughout the paper. In a $M/M/1$ queue, jobs of class c arrive to their queue c according to a Poisson process of rate $\lambda_c > 0$. If the server of queue $c \in \mathcal{C}$ uses power p, that queue's processing time for each job is exponential with rate $\mu_c(p)$. The response/sojourn time is $R_c(p) = \frac{1}{\mu_c(p) - \lambda_c}$, which sets the minimum power level to $\epsilon_c = [\mu_c]^{-1}(\lambda_c)$. The rate $\mu_c(p)$ is strictly increasing in p and is determined by DVFS as follows: the speed μ vs. power p dependence in computer processors is typically of the form $p = \eta\mu^\delta + \xi$ with $\eta > 0$, $\xi \geq 0$ and $\delta \in [2, 3]$ [18], [19]. For example, if $\eta = 1$ and $\xi = 0$, the response time of queue $c \in \mathcal{C}$ is

$$R_c(p) = \frac{1}{p^{(1/\delta_c)} - \lambda_c}. \tag{1}$$

Remark 1. The response time is nonlinear. If queue c uses an insufficient amount of power, the system is unstable and the response time $R_c(p)$ would explode to infinity (leaving nothing to optimize). The paper considers conditions to prevent $R_c(p)$ from exploding.

A. Quality-of-Service Targets and Performance Loss Cost

Each class $c \in \mathcal{C}$ has a quality-of-service or performance target, which is expressed as an average response time $\overline{R}_c$ that jobs of class c want to attain. We let $\overline{R} = \sum_{c \in \mathcal{C}} \overline{R}_c$ be the sum of response times. Any overshoot $[R_c(p) - \overline{R}_c]^+$ of the response time target $\overline{R}_c$ caused by queue $c \in \mathcal{C}$ using (insufficient) power p leads to a performance loss cost

$$l_c(p) = \alpha_c \left([R_c(p) - \overline{R}_c]^+\right)^\gamma. \tag{2}$$

In (2), $\alpha_c > 0$ is a weighting factor, and $\gamma > 1$ reflects how 'aggressive' the cost is to response time overshoot. Since $R_c(p)$ is strictly decreasing in p, there is a unique target power level $\overline{p}_c$ that achieves the performance target: $\overline{R}_c = R_c(\overline{p}_c)$. Observe $\overline{p}_c = R_c^{-1}(\overline{R}_c)$, with performance loss cost $l_c(\overline{p}_c) = 0$. We define the total target power as $\overline{P} = \sum_c \overline{p}_c$. As an example, for the $M/M/1$ response time given in (1), the target power is $\overline{p}_c = \left(\lambda_c + \frac{1}{\overline{R}_c}\right)^{\delta_c}$. The derivative of the performance loss (2) with respect to power is

$$l_c'(p) = \alpha_c \gamma R_c'(p) \left(R_c(p) - \overline{R}_c\right)^{\gamma - 1} \tag{3}$$

when $p_c \leq \overline{p}_c$, and 0 otherwise.

Remark 2 (Properties of Performance Loss $l_c(p)$ function)**.** Observe that the performance loss cost (2) has the following properties (recalling that $\gamma > 1$):

1) $l_c(p) = \infty$ for $p \in [0, \epsilon_c]$ and $l_c(p) = 0$ for $p \geq [\overline{p}_c, \infty)$.
2) $l_c(p) > 0$ and is strictly decreasing for $p \in [\epsilon_c, \overline{p}_c]$.
3) Since $\gamma > 1$, Equation (3) gives $l_c'(\overline{p}_c^+) = 0 = l_c'(\overline{p}_c^-)$, so $l_c(p)$ is differentiable at $\overline{p}_c$ (and hence, differentiable everywhere) and in particular $l_c'(\overline{p}) = 0$. ■

The later results (in Proposition 1) leverage the properties in Remark 2. Next, we make two assumptions about the performance loss function $l_c(p)$.

Assumption 1. Assume that $l_c(p)$ is strictly convex over $(\epsilon_c, \overline{p}_c)$, i.e., $l_c''(p) > 0$. ■

Assumption 1 is used to prove Proposition 1, and its combination with the next assumption is used to prove convergence results in Section IV.

Assumption 2. Assume $l_c'''(p) < 0$ for $p \in (\epsilon_c, \overline{p}_c)$.

It can be verified that the M/M/1 queue response time (1) with $\gamma = 2$ in (2) satisfies Assumption 1 and Assumption 2.

The system is incentivized to draw more power to reduce the performance loss l_c, a decreasing function of power. The next subsection introduces a cost to push back against drawing too much power, formulated as a power price.

B. Push-Back Pressure and Power Cost

To disincentivize using power excessively or frivolously, the power management algorithm imposes a "push-back" power cost. This power cost is an internal pricing signal sent by the system operator to its servers, aiming to manage the performance versus power trade-off.

Let $P = \sum_{c \in \mathcal{C}} p_c$ be the total power consumption. The power price π is a function of the total power used by all queues: $\pi = \Pi(P)$, where the pricing function $\Pi(\cdot)$ has the following natural properties:

- $\Pi(P) > 0$ and is is twice differentiable for $P \in (0, \infty)$.
- $\Pi(P)$ is strictly increasing ($\Pi'(P) > 0$) for $P \in (0, \infty)$, reflecting that the higher the usage and demand for power, the more expensive it becomes.

For concreteness, consider a pricing function of the form

$$\Pi(P) = P^{\beta}, \text{ where } \beta > 0. \tag{4}$$

Π is strictly convex (concave) when $\beta > 1$ ($\beta < 1$). A strictly convex price $\Pi(P)$ models an aggressively managed system, where the power price's increase rate gets larger as more power is consumed. Such a pricing system may reflect an uncooperative setting such as an electric grid, where the grid operator aims to aggressively regulate independent, greedy users that work for their own self-interest. On the other hand, a data center is a "cooperative" setting where the system manager controls all the servers. In other words, all queues share the same goal of managing power efficiently for the data center as a whole. This motivates using a concave function for determining a power price that is internal to the data center. Concavity reflects that the system is "naturally" well-regulated: the system only consumes high amounts of power when it sufficiently needs it, and therefore high power usage should be accommodated by slowing down the rate of price increase. Indeed, when $\beta < 1$, the power management scheme discussed below is proven to be stable. Exploring a convex power cost that reflects the actual market price of electricity set by an external power provider, for example, in dollars per Watt, is an area for future work.

We assume the response time $R_c(p)$ and performance loss costs are a continuous function of p. Discrete-valued costs with different levels of performance is also an area of future work.

Remark 3. The parameter choices for α_c, γ, β are application-specific. The parameters γ, β reflect how aggressively the system designer penalizes the overshoot penalty and applies it as negative feedback. The parameter α_c reflects how much the system designer values performance versus power. Section V also discusses convergence effects of γ, β.

The previous two subsections demonstrate a trade-off between the performance loss cost $l_c(p)$ that is decreasing and the power cost πp that is increasing with power. The next subsection defines an algorithm to balance this trade-off over time.

C. Power Management Algorithm and System Evolution

The algorithm can be formulated as a discrete-time dynamical system that determines the power consumption, as follows. The system evolves in phases or periods indexed by $t = 1, 2, 3, \cdots$. Let p_c^t be the power used by queue $c \in \mathcal{C}$ at time t, and let $p^t = (p_c^t, c \in \mathcal{C}) \in \mathbb{R}_+^C$ be its vector form. To start the algorithm, in the first period $t = 1$, the system manager sets a price $\pi^1 \in (0, \infty)$ (perhaps even arbitrarily or randomly) and broadcasts it to the queues. Each queue $c \in \mathcal{C}$ responds by selecting powers p_c^1. This determines the next period's price $\pi^2 = \Pi\left(\sum_{c \in \mathcal{C}} p_c^1\right)$. This induces the queues' next powers p_c^2, and so on, according to the following iteration. Given the power price π during a period, each queue $c \in \mathcal{C}$ autonomously (and selfishly) chooses its power to minimize its own total cost during that period,

$$k_{c,\pi}(p) = l_c(p) + \pi p, \tag{5}$$

comprised of the performance loss cost $l_c(p)$ and the power cost πp. Let $\Phi_c(\pi) = \arg\min_p k_{c,\pi}(p)$. At time t, if the power price is π^t then queue $c \in \mathcal{C}$ uses power

$$p_c^t = \Phi_c(\pi^t) = \arg\min_p \left\{ \alpha_c \left([R_c(p) - \overline{R}_c]^+ \right)^{\gamma} + \pi^t p \right\}. \tag{6}$$

Note $p_c^t = -l_c^*(-\pi^t)$, where l_c^* is the convex conjugate function of l_c [20]. The weight α_c reflects the sensitivity of job class c to response performance loss vs. power. Writing the vector $\Phi(\pi) = (\Phi_c(\pi), c \in \mathcal{C})$, the vector of powers chosen at period t is $p^t = \Phi(\pi^t)$. Let $P^t = \sum_{c \in \mathcal{C}} p_c^t$ be the *aggregate* power consumption throughout period t. After the queues choose power p^t at period t according to (6), the power manager observes their total P^t and sets the power price π^{t+1} for the next period $t + 1$ as

$$\pi^{t+1} = \Pi(P^t), \tag{7}$$

which it then broadcasts to all queues. Let $F_c = \Phi_c \circ \Pi$. Combining (6) and (7) gives for every $c \in \mathcal{C}$,

$$p_c^t = \Phi_c\left(\Pi\left(\sum_{c \in \mathcal{C}} p_c^{t-1}\right)\right) = F_c(P^{t-1}). \tag{8}$$

Define $F = \sum_c F_c = \sum_c \Phi_c \circ \Pi$. Summing (8) over $c \in \mathcal{C}$, the total power P^t evolves as $P^{t+1} = F(P^t)$. For the system to be stable and therefore practically implementable, for every $c \in \mathcal{C}$, p_c^t must converge as $t \to \infty$. Therefore, we are primarily interested in the convergence of p_c^t. Since $p_c^{t+1} = \Phi_c(P^t)$ by (8) and $P^t = \sum_c p_c^t$, the powers p_c^t converges for every $c \in \mathcal{C}$ if and only if P^t converges.

Remark 4 (Separation of timescales)**.** In our analysis, we assume that parameters such as the arrival rate λ_c are constant over time. In practice, if they are unknown or vary over time, they can be learned/estimated online. We assume that 1) the timescale for the queues to converge to their steady states (and for estimation of steady state response) is significantly smaller than 2) the timescale of convergence of the power management algorithm P^t and 3) the latter is significantly smaller than the timescale of variations of parameters (like average average arrival rates). Thus, there is significant separation of timescales.

Remark 5. In practice, to implement the power control algorithm, the power values are transmitted as quantized values. For our analysis, we assume that the communication bandwidth between the queues and the system manager can support enough bits to transmit the power values, so that the power can be approximated as having continuous values. In addition, in practice, the queues are of finite length, but we assume they are long enough so that they can be approximated as infinite (i.e. they are oveflowing with very low probability).

Because of the many characteristics that may distinguish a class, there can be many classes. The response time R_c quantifies the quality of experience for each class c as a function of power. Per period, the power manager observes only the *aggregate* power consumption P across all queues and broadcasts only a *single price* value to all queues. Hence, the system requires minimal communication can scale up to large populations of queues. Some more useful system properties are summarized below.

Proposition 1. Under the assumptions given before on the price $\Pi(\cdot)$, and Assumption 1, for every queue $c \in \mathcal{C}$:

1) For $\pi \in (0, \infty)$, Equation (5) admits a unique minimizer p_c given by $p_c = \Phi_c(\pi) = [l_c']^{-1}(-\pi) \in (\epsilon_c, \overline{p}_c)$.
2) Let $P^{\text{opt}} = \sum_c p_c$ be the aggregate optimal power, where p_c is as defined above. Then $P^{\text{opt}} \in (P_\epsilon, \overline{P})$.
3) The minimizer $p_c = \Phi_c(\pi)$ of (5) is a continuous and strictly decreasing function of π.
4) $F_c(P)$ and $F(P)$ are continuous and strictly decreasing for $P \geq 0$.
5) When P^{opt} is close enough to P_ϵ (denoted by $P^{\text{opt}} \approx P_\epsilon$), $F(P^{\text{opt}}) > P^{\text{opt}}$. When P^{opt} is close enough to $\overline{P}$ (denoted by $P^{\text{opt}} \approx \overline{P}$), $F(P^{\text{opt}}) < P^{\text{opt}}$.

Proof sketch. By convexity of l_c (Assumption 1) and Property (3) in Remark 2, $l_c'(p)$ strictly decreases from ∞ at $p = \epsilon_c$ down to 0 for $p \geq \overline{p}_c$. Therefore, setting the derivative of the cost (6) to zero gives the equation $l_c'(p) = -\pi < 0$, which has a unique solution $p_c \in (\epsilon_c, \overline{p}_c)$. This proves (1). Then, (2) follows directly from (1). Property (3) also follows from (1), together with l_c' being continuous and strictly decreasing (given by Remark 2). Next, (3) and $\Pi(\cdot)$ being strictly increasing and continuous gives (4). For (5), when $P^{\text{opt}} \approx P_\epsilon$, as the optimal powers $p_c^t \to \epsilon_c$, $l_c'(p_c^t) \to -\infty$ and so eventually it holds that $p_c^t < [l_c']^{-1}(-\Pi(P^{\text{opt}})) = p_c^{t+1}$. On the other hand when $P^{\text{opt}} \approx \overline{P}$, as $p_c^t \to \overline{p}_c$, $l_c'(p_c^t) \to 0$ and so eventually $p_c^t > [l_c']^{-1}(-\Pi(P^{\text{opt}})) = p_c^{t+1}$. Summing over $c \in \mathcal{C}$ gives the result. ☐

As an example, consider M/M/1 queues with response time (1). Suppose all C queues/classes are identical with $\lambda_c = \lambda, \alpha_c = 1, \delta_c = 2, \overline{R}_c = 1$ for all $c \in \mathcal{C}$. Then $\epsilon_c = \lambda^2$ and the target response time is attained for $\overline{p}_c = (1+\lambda)^2$. The performance loss vs. power cost (5) minimized by each queue $c \in \mathcal{C}$ over $p \in (\epsilon_c, \overline{p}_c)$ given the price π is

$$k_{c,\pi}(p) = \left(\frac{1}{\sqrt{p} - \lambda} - 1\right)^\gamma + \pi p. \tag{9}$$

Because the queues are identical, the system is "symmetric" and all queues $c \in \mathcal{C}$ will have the same minimizer p_c of (9). Therefore, $P^{\text{opt}} = Cp_1$ is the aggregate power of the queues, while $P_\epsilon = C\lambda^2$ and $\overline{P} = C(1+\lambda)^2$. For example, if $\lambda = 1$ then by Property (2) of Proposition 1, the total power consumption P ranges between $P_\epsilon = C$ and $\overline{P} = 4C$, scaling linearly with the number of queues.

In general, Property (1) in Proposition 1 yields the following. First, since the minimizer of (5) is unique, (6) is well-defined. Second, if l_c is simple enough that a closed-form expression for $[l_c']^{-1}$ exists, then $p_c^t = [l_c']^{-1}(-\pi^t)$ can be easily found. Otherwise, gradient descent can be used to solve (6) and compute p_c^t. Third, at every time period t, the power used is bounded as $p_c^t \in (\epsilon_c, \overline{p}_c)$ and $P^t \in \left(P_\epsilon, \overline{P}\right)$. Thus in practice, F and F_c will only take in $P \in (P_\epsilon, \overline{P})$, although $F(P), F_c(P)$ are still well-defined for all $P \geq 0$.

Note that as $p_c^t > \epsilon_c$, the algorithm ensures finite response time for all t. Since $p_c^t < \overline{p}_c$, the response times will exceed their target: $\overline{R}_c(p_c^t) > \overline{R}_c$. The target response time mandates a lower bound that delineates priority between different queues: a less urgent queue will tolerate a larger "minimum" delay $\overline{R}_c < \overline{R}_c(p_c^t)$ for jobs to finish, allowing more power to be allotted to higher priority, faster queues. The power cost dampens the response time from overshooting its target too much and causing the performance loss to explode over time. Instead, the power trajectory P^t converges to a steady state value, as we next prove. First, we describe some key lemmas about fixed points.

III. Helper Lemmas for Convergence Proof

Recall that x^* is a *fixed point* of a function g if $g(x^*) = x^*$. In this section we state some results about fixed points that will be useful in showing convergence of P^t.

Remark 6. If $g : \mathbb{R} \to \mathbb{R}$ is nondecreasing and bounded then for any x, $g^n(x)$ is a monotone, bounded sequence that converges to some x^*. If g is also continuous, $g(x^*) = g(\lim_{n\to\infty} g^n(x)) = \lim_{n\to\infty} g(g^n(x)) = \lim_{n\to\infty} g^{n+1}(x) = x^*$, so x^* is a fixed point of g.

Remark 6 will be used to show that the even and odd subsequences of P^t converge, as described next.

Lemma 1. For any initial aggregate power P^1, the limits of P^{2t} and P^{2t+1} exist, and are fixed points of F^2.

Proof sketch. By Property (4) of Proposition 1, F is continuous and nonincreasing. Therefore F^2 is continuous and nondecreasing. From any initial state P^1, for all $t \geq 1$, $P^t \in [0, \overline{P}]$. Applying Remark 6 with $g = F^2, x = P^1$ and with $g = F^2, x = F(P^1)$ gives the result. ☐

Lemma 1 suggests that the even and odd subsequence limits of P^t will coincide if F^2 has only one fixed point.

Lemma 2. If F^2 has a unique fixed point P^*, then for any initial state $P^1 > 0$, $\lim_{t\to\infty} P^t = P^*$.

Proof sketch. By Lemma 2, P^{2t} and P^{2t+1} converge to fixed points of F^2, but F^2 has only one fixed point.

So $\lim_{t\to\infty} F^{2t}(P^0) = P^* = \lim_{t\to\infty} F^{2t}(F(P^0)) = \lim_{t\to\infty} F^{2t+1}(P^0)$. So $P^t \to P^*$. $\square$

The next lemma relates the fixed points of F and F^2.

Lemma 3. F has a unique fixed point P^*, and P^* is also a fixed point of F^2.

Proof sketch. By Property (4) in Proposition 1, F is strictly decreasing. Also, $F(0) = \overline{P}$. So, $g(P) = F(P) - P$ is strictly decreasing and continuous, with $g(0) > 0$. So $g(P) - P = 0$ for exactly one P. Therefore F has a unique fixed point P^* at which $g(P^*) = 0$. And P^* is a fixed point of F^2 since $F^2(P^*) = F(F(P^*)) = F(P^*) = P^*$. $\square$

Since a unique fixed point of F^2 guarantees convergence, the next step is to find conditions that ensure this fixed point.

Lemma 4. If F is strictly convex, then F^2 has a unique fixed point.

Proof sketch. Let P^* be the fixed point of F and F^2 as described in Lemma 3. Let $a = P^*$. Suppose to the contrary that F^2 has another fixed point $b \neq a$. If $b < a$ then letting $d = F(b) > F(a) = a$ gives $F^2(d) = F(F^2(b)) = F(b) = d$, so $d > a$ is another fixed point of F^2. So, without loss of generality, assume $b > a$. Since F is strictly decreasing and strictly convex, so is F^{-1}. For any fixed point d of F^2, observe that $F(d) = F^{-1}(d)$. Therefore, $b < \overline{P}$ since otherwise, $F^{-1}(b) \leq F^{-1}(\overline{P}) = 0$ but $F(b) > 0$ so $F(b) \neq F^{-1}(b)$. So for some $\theta \in (0,1), b = \theta a + (1-\theta)\overline{P}$.

We will split into two cases: $a \leq \overline{P}$ and $a > \overline{P}$. If $a \leq \overline{P}$ we will use the tangent inequality for differentiable, strictly convex functions g: for any $x_1 \neq x_2$, $g(x_2) > g'(x_1)(x_2 - x_1)$. If $a > \overline{P}$ we will use the secant inequality for strictly convex functions g: For any $x_1 < x_2 < x_3$, $\frac{g(x_2)-g(x_1)}{x_2-x_1} < \frac{g(x_3)-g(x_1)}{x_3-x_1}$.

Case 1 : First suppose $a \leq \overline{P}/2$. Since F^{-1} is strictly convex, $F(b) = F^{-1}(b) = F^{-1}(\theta a + (1-\theta)\overline{P}) < \theta F^{-1}(a) + (1-\theta)F^{-1}(\overline{P}) = \theta a$. We will reach a contradiction by showing that $\theta a < F(b)$, as follows. Since F is convex, $F(b) \geq F(a) + F'(a)(b-a) = a + F'(a)(b-a)$. Plugging in $b = \theta a + (1-\theta)\overline{P}$, we get $F(b) - \theta a > (1-\theta)a + F'(a)(b-a) = (1-\theta)(a + (\overline{P}-a)F'(a))$, which we now show is positive. Since F is strictly convex, by the tangent inequality, $\overline{P} = F(0) > F(a) + (a-0)F'(a)$. So $F'(a) > \frac{a-\overline{P}}{a}$. Therefore $a + (\overline{P}-a)F'(a) > a + (\overline{P}-a)\frac{(a-\overline{P})}{a} = \frac{1}{a}(a^2 - (a-\overline{P})^2) = \frac{\overline{P}}{a}(\overline{P} - 2a) \geq 0$, using $a \leq \overline{P}/2$. Hence, $F(b) > \theta a$, a contradiction.

Case 2: Suppose $a < \overline{P}/2$. The secant inequality applied to $a < b < \overline{P}$ makes $\frac{F(b)-a}{b-a} = \frac{F^{-1}(b)-a}{b-a} = \frac{F^{-1}(b)-F^{-1}(a)}{b-a} < \frac{F^{-1}(\overline{P})-F^{-1}(a)}{\overline{P}-a} = -\frac{a}{\overline{P}-a}$. So $\frac{a-F(b)}{b-a} > \frac{a}{\overline{P}-a} \geq \frac{\overline{P}/2}{1-\overline{P}/2} = 1$, using $a \geq \overline{P}/2$. Therefore $a - F(b) > b - a$. So $a > \frac{1}{2}F(b) + \frac{1}{2}b$. On the other hand, since F is strictly decreasing and strictly convex, $a = F(a) < F(\frac{1}{2}F(b) + \frac{1}{2}b) < \frac{1}{2}F^2(b) + \frac{1}{2}F(b) = \frac{1}{2}b + \frac{1}{2}F(b)$, a contradiction. $\square$

We note that Lemma 4 clarifies and rigorizes the sketch of the argument about the (potentially) unique fixed point property of F^2 used in [17] to argue for system convergence.

The next section uses the above lemmas to prove convergence of P^t.

IV. System Convergence Proof

Lemma 5. If Assumption 1 and Assumption 2 hold and $\Pi(P)$ is concave, F_c is convex on $P \in [0, \overline{P}]$.

Proof sketch. By Property (1) in Proposition 1, $F_c(P) = [l_c']^{-1}(-\Pi(P))$. Let $y = F_c(P)$. So for $P > 0$, $F_c'(P) = -([l_c']^{-1})'(-\Pi(P))\Pi'(P) = -\frac{\Pi'(P)}{l_c''(y)}$. Using the standard quotient rule for derivatives, the numerator of $-F_c''(P)$ is, $\Pi''(P)l_c''(y) + \frac{l_c'''(y)}{l_c''(y)}\Pi'(P)^2$, which is negative for $P > 0$ under our assumptions. Therefore $F_c(P)$ is strictly convex for $P > 0$. Since F_c is strictly decreasing for $P \geq 0$ by Property (4) of Proposition 1, it can be shown that F_c is also strictly convex at $P = 0$. $\square$

Convexity of F_c will be key in establishing convergence.

Lemma 6. If F_c is convex on $[0, \overline{P}]$ for all $c \in \mathcal{C}$, then P^t converges to the fixed point of F^2.

Proof sketch. Since F_c is convex on $[0, \overline{P}]$ for all $c \in \mathcal{C}$, F is convex on $[0, \overline{P}]$. By Lemma 4, F^2 has a unique fixed point. Applying Lemma 2 gives the result. $\square$

Finally, the next result gives convergence of P^t.

Proposition 2. Under Assumption 1, Assumption 2 and a pricing function $\Pi(P) = P^\beta$ given in (4) with $\beta < 1$, P^t converges to the fixed point P^* of F^2, i.e., $P^t \xrightarrow{t\to\infty} P^*$.

Proof sketch. Since $\beta < 1$, $\Pi(P)$ is concave. Therefore we can apply Lemma 5, and then apply Lemma 6. $\square$

The next theorem describes the convergence of the system.

Theorem 1. Under Assumption 1 and Assumption 2 (e.g. M/M/1 queue response time (1) with $\gamma = 2$ in (2)), and pricing function $\Pi(P) = P^\beta$ (4) with $\beta < 1$, the power management system satisfies the following properties:

1) F has a unique fixed point $P^* = F(P^*)$ with $F(P) > P$ for $P \in (P_\epsilon, P^*)$ and $F(P) < P$ for $P \in (P^*, \overline{P})$.
2) The aggregate power converges to P^* as $t \to \infty$.
3) The prices π^t converge to $\pi^* = \Pi(P^*)$.
4) The power in each queue converges to $p_c^* = F_c(P^*)$.

Proof sketch. Property 1) follows from Lemma 3 and the property that F is strictly decreasing, as shown in Proposition 1. Proposition 2 gives 2). Property 3) holds from continuity of Π. Continuity of F_c and (8) give 4). $\square$

Property (4) of Theorem 1 implies that the system converges to optimal powers p_c^* and a price π^* satisfying the equation $p_c^* = \arg\min_p\{l_c(p) + \pi^* p\}$ for every $c \in \mathcal{C}$. The algorithm arrives over time at a power usage that strikes a balance between the power cost and performance loss. For all t, $p_c^t \leq \overline{p}_c$ by Property (1) of Proposition 1, which gives the bound $p_c^* \leq \overline{p}_c$.

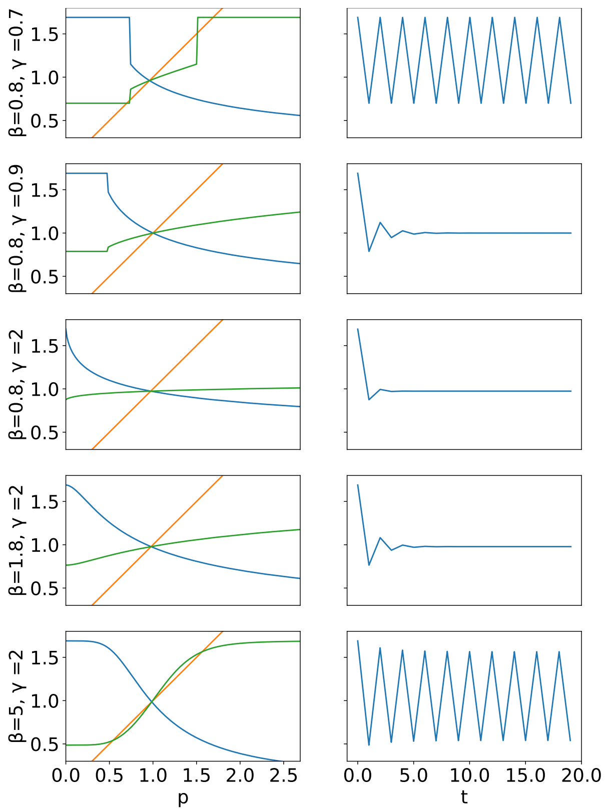

Figure 1. Power management simulation for $C = 1$ class, $\lambda = 0.3$ and $\overline{R} = 1$. Left column: $F(P)$ (blue) and $F^2(P)$ (green) are plotted. They intersect the orange line $g(P) = P$ at their fixed points. Right column: P^t illustrating convergence or divergence.

Lemma 5 suggests that Theorem 1 also holds for more general, reasonable pricing functions beyond (4) that are strictly increasing, twice differentiable and concave. Convergence analysis for $\beta > 1$ will be analyzed in future work. We also note that in this model, the queues are synchronized to update their power the same time periods. Asynchronous convergence of the system will be explored in future work.

V. Performance Analysis

This section uses numerical experiments to analyze performance. First, Section V-A verifies system stability under the conditions given in Theorem 1 for a simple example, and gives example of oscillatory behavior when those conditions are violated. Then, when the power consumption does converge to a limit, we analyze its limit in Section V-B as a function of the parameters α and $\overline{R}$. Finally, Section V-C demonstrates that the system can handle a large population of queues, including random arrivals and removals of queues.

A. *Stability versus Oscillation Analysis*

We simulate the simple $M/M/1$ example with total cost (9), arrival rate $\lambda = 0.3$, and $C = 1$ class. The simplicity of this example allows isolating the effects of the pricing and performance loss exponents, β and γ, respectively, and examining the relationship between stability and fixed points. The result of a simulation with a different pair of (β, γ) is plotted in each row of Figure 1. The left column of Figure 1 plots F in blue and F^2 in green. The right column plots the trajectory P^t. In all plots in the left column, F intersects the orange line $g(P) = P$ at a single fixed point, and F^2 also intersects and shares this fixed point. This is consistent with Lemma 3. In the middle three rows, F^2 only intersects the orange line at the point where F does, demonstrating that F^2 has a unique fixed point, and P^t converges to that point, as seen in the right column. However, in the first and last row, F^2 intersects the orange line at multiple other fixed points, and P^t oscillates. This demonstrates Lemma 2.

In the bottom three rows, $\gamma = 2$. When $\beta < 1$ in the third row, the blue curve F is convex (and P^t converges, demonstrating Theorem 1), but not when $\beta > 1$, as consistent with Lemma 5. As β increases from the middle to the bottom row, we see that F becomes more concave, making the green curve F^2 "rotate counterclockwise" until in the bottom row, it hits the orange line at multiple fixed points that P^t oscillates between. In the fourth row, even though $\beta > 1$, P^t still converges, showing that the sufficient condition in Proposition 2 is not necessary for convergence, which will be further analyzed in future work.

Next, the significance of the $\gamma > 1$ assumption is analyzed by reducing γ to be under 1 in the first two rows of Figure 1. We see that this makes F initially flat, breaking convexity. When γ is only slightly less than 1, P^t still converges, as seen in the second row. However, when γ is too small, P^t oscillates, as seen in the first row. Thus, γ being too small β being too large can induce oscillations in the system. The system engineer must carefully design system parameters to ensure stability, e.g. following a rule of thumb that $\gamma > 1$ (e.g. $\gamma = 2$) and $\beta < 1$.

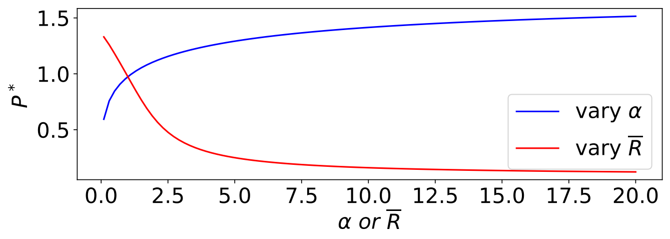

Figure 2. The power P^* that the system converges to is plotted as a function of α and $\overline{R}$. In the blue curve, α is varied as $\overline{R} = 1$. In the red curve, $\overline{R}$ is varied as $\alpha = 1$. There is $C = 1$ class, $\lambda = 0.3, \gamma = 2,$ and $\beta = 0.8$.

B. *Trade-off Between Weighting Parameter and Target Time*

Here, $\beta = 0.8 < 1$ and $\gamma = 2$ are fixed so that Proposition 2 guarantees the power P^t converges to a limit P^*. Figure 2 then plots the value of P^* as a function of the weighting factor α (in blue) and the target response time $\overline{R}$ (in red). As α increases, the queues place more weight on minimizing the performance loss l_c, so they use more power, as shown by the increasing blue curve. As $\overline{R}$ increases, the jobs can tolerate more latency as the target response time becomes longer, so the queues need less power, as shown by the decreasing red curve.

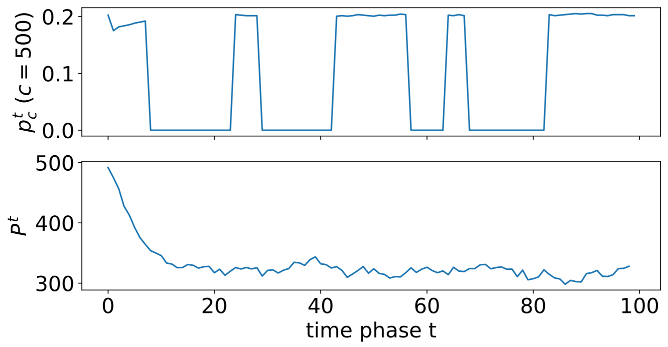

Figure 3. System evolution with $C = 1000$ queues. We set $\gamma = 2, \beta = 0.8$, and $\lambda_c, \alpha_c, \overline{p}_c \sim \text{Unif}(0.1, 1)$. At any time, each queue independently enters or leaves the system with 5% probability. Top plot: the trajectory of p_c^t for an arbitrarily chosen queue ($c = 500$). Bottom plot: the aggregate power P^t over timer period t.

C. Robustness in Time-Varying, Large Systems

Now, we increase the number of queues to $C = 1000$, where each queue follows the $M/M/1$ response time (1) with $\delta_c = 2$. Each queue is independently and randomly assigned $\lambda_c, \alpha_c, \overline{p}_c$ according to a uniform distribution over $(0.1, 1)$. We set $\beta = 0.8, \gamma = 2$ to ensure stability by Proposition 2. In a data center, queues may leave the system and re-enter, so the number of queues is time-varying. This can be caused by random server crashes or idle periods in the network. To model this, we assign a "on" or "off" status to every queue. We initially set all 1000 queues to "on," i.e. all being in the system. At each time period t, every queue has a 5% probability of changing its status: i.e. deactivating and leaving the system if it is in the system ("on" to "off"), or activating and entering if it is not ("off" to "on"). In the bottom plot, the aggregate power P^t stabilizes after $t = 20$ time phases. The top plot shows the trajectory of a particular queue, $c = 500$, chosen arbitrarily. After an initial transitory period, the power level is stable during the periods that the queue is awake. The system demonstrates fault tolerance and stability. Further quantification of robustness and simulation of data center applications is analysis for future work.

VI. Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1656518.

References

[1] D. Mytton and M. Ashtine, "Sources of data center energy estimates: A comprehensive review," *Joule*, vol. 6, no. 9, p. 2032–56, 2022. [Online]. Available: https://www.cell.com/joule/pdf/S2542-4351(22)00358-0.pdf

[2] A. S. G. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015. [Online]. Available: https://www.mdpi.com/2078-1547/6/1/117

[3] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[4] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aba3758

[5] I. RatkoviÄ, N. BeÅŸaniÄ, O. S. Ãnsal, A. Cristal, and V. MilutinoviÄ, "Chapter one - an overview of architecture-level power- and energy-efficient design techniques," ser. Advances in Computers, A. R. Hurson, Ed. Elsevier, 2015, vol. 98, pp. 1–57. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245815000303

[6] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *37th IEEE International Conference on Solid-State Circuits*, Feb. 1990.

[7] M. Weiser, B. Welch, A. Demers, and S. Shenker, *Scheduling for Reduced CPU Energy*. Boston, MA: Springer US, 1996, pp. 449–471. [Online]. Available: https://doi.org/10.1007/978-0-585-29603-6_17

[8] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithm for dynamic speed-setting of a low-power CPU," in *Proceedings of the 1st annual international conference on Mobile computing and networking*, ser. MobiCom '95. New York, NY, USA: Association for Computing Machinery, Dec. 1995, pp. 13–25.

[9] S. Albers and H. Fujiwara, "Energy-efficient algorithms for flow time minimization," *ACM Trans. Algorithms*, vol. 3, no. 4, Nov. 2007.

[10] N. Bansal, K. Pruhs, and C. Stein, "Speed scaling for weighted flow time," *SIAM Journal on Computing*, vol. 39, no. 4, pp. 1294–1308, 2010.

[11] T.-W. Lam, L.-K. Lee, I. K. To, and P. W. Wong, "Speed scaling functions for flow time scheduling based on active job count," in *Algorithms-ESA 2008: 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings 16*. Springer, 2008, pp. 647–659.

[12] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH computer architecture news*, vol. 35, no. 2, pp. 13–23, 2007.

[13] M. Ghamkhari and H. Mohsenian-Rad, "Energy and performance management of green data centers: A profit maximization approach," *IEEE transactions on Smart Grid*, vol. 4, no. 2, pp. 1017–1025, 2013.

[14] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," *ACM SIGOPS operating systems review*, vol. 35, no. 5, pp. 103–116, 2001.

[15] B. N. BACCELLI, F. and N. GAST, "Distributed delay-power control algorithms for bandwidth sharing in wireless networks," *IEEE/ACM Trans. on Networking*, vol. 19, no. 5, 2011.

[16] A. R. Khan, A. Mahmood, A. Safdar, Z. Khan, and N. Khan, "Load forecasting, dynamic pricing and dsm in smart grid: A review," *Renewable and Sustainable Energy Reviews*, 01 2015.

[17] N. Master, J. Mounzer, and N. Bambos, "Distributed smart grid architecture for delay and price sensitive power management," in *2014 IEEE International Conference on Communications (ICC)*, 2014, pp. 3670–3675.

[18] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices," in *Int. Conf. Parallel Processes Appl. Math.*, 2013.

[19] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar, "Critical power slope: understanding the runtime effects of frequency scaling." 01 2002, pp. 35–44.

[20] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.