Scalable Data Center Power Management via a Global Stress Signal

Daniel Miller, Neal Master, Zhengyuan Zhou and Nicholas Bambos
Department of Electrical Engineering
Stanford University, Stanford, CA 94305
Email: {danielrm, nmaster, zyzhou, bambos}@stanford.edu

Abstract—In this paper, we develop a general-use autonomous control strategy for managing the trade-off between processing throughput and power consumption in data centers. This approach relies on the concept that the delay in completing computational tasks can be reduced at the cost of more power. The scheme's generality allows it to be applied at multiple hierarchical levels within a data center, or another system with similar architecture. In particular, we show that our scheme converges asynchronously to a unique solution. This property allows the control strategy to be implemented in a low-complexity, yet robust and scalable manner. These properties are particularly important when considering data center power control system architectures, which can involve a wide variety of distributed computing resources performing diverse tasks. The presented scheme is mostly decentralized, except for a single global power stress signal provided by a redundant central authority. Based on this power stress, computing resources independently and autonomously manage power consumption to optimally balance power versus delay.

I. INTRODUCTION

Large-scale computing data centers expend significant power to perform computational tasks in a timely manner. Optimizing the power/delay trade-off is a necessary goal of data center power management. This problem is already large, and will only continue to grow. Advances in computing technology and in large-scale information analysis and manipulation have caused a societal convergence to the modern emphasis on, and demand for large-scale computing. Data centers service this demand in localized and closed, yet diverse ecosystems consisting of a variety of computational resources. Efficiency, robustness, and scalability are significant concerns in the design and operation of such data centers [1][2][3]. Additionally, these centers must be able to appropriately handle the unique nature of different types of information and computational tasks. Not all bits are created equal; different forms of information can possess their own unique requirements, such as time-sensitivity or accuracy versus energy consumption [4]. Such distinctions are also true for the computational services provided by data centers. A major portion of a data center's operational cost is its energy usage, both for powering the computing processors and associated infrastructure resources, and for temperature regulation [5]. This energy cost is already significant, and is only expected to increase in the future. Electricity used by data centers worldwide doubled from 2000 to 2005, and increased by about 56% from 2005 to 2010 [6]. Aside from the evident economic incentives for increasing efficiency, motivations include serious ecological considerations ranging from CO₂ production to the use and disposal of e-waste materials [7][8]. These issues can be at least partially alleviated by considering demand-response strategies. For instance, load shaping strategies like peak shifting allow for equivalent overall energy usage at a lower cost and reduced system stress, thus increasing system lifetime. The data center's robustness and scalability are two additional concerns which must be taken into account. It is natural to address rising demand through expansions to an existing data center [9]. Partially due to such expansions, operating costs of large-scale data centers are increasing by an average of 20% per year [10].

To address these issues, we consider a particular distributed control architecture. Departing from centralized control enhances the data center's robustness by avoiding a hierarchy based on a single potential point of failure, and facilitates scalability, particularly when considering autonomously behaving resources. [11]. In our proposed architecture, control responsibility is inherited by discrete collections of the data center's computing resources. Each so-called resource collection is capable of solving a particular class of computational services provided by the data center. For example: a resource collection could range from a rack of GPUs, to a single multi-core processor. Incoming tasks to a particular resource collection are considered in a relatively standard manner using a queue model, each with an associated delay sensitivity function [12]. The time required to complete an individual task—and thus the queue service rate—is connected to the power that the resource collection consumes. This relationship holds even for co-dependent tasks in a blocking resource, so long as all connected tasks are considered within a single resource collection. The power-time relationship can be achieved by selectively powering on cores to solve more problems simultaneously, or by frequency scaling the cores to solve problems more quickly, or by a number of other methods [13]. For simplicity, we consider resource collections that support a continuous power consumption range, for instance by tolerating sufficiently wide ranges of operating frequencies, and possessing a large enough number of select-able cores. Every collection in the data center may therefore service the computational tasks in its queue at a controllable rate.

Our control problem now possesses notable similarities to standard problems in other fields. For example, wireless communication architectures often use distributed power control to compensate for the effects of interference [14]. In this context, users can autonomously vary data throughput, and thus transmitter power, according to the locally measured interference level. The particular implementation presented in this paper was inspired by a line of thinking in the context
of smart grid power control [15]. By controlling a global price signal, a central overseer can encourage consumers to modify their power usage according to their own valuation of price versus delay. Such behavior most often comes into play in the form of demand response and load shaping. To solve this problem in the data center context, we introduce the concept of a global power stress level. This variable provides a measure of the overall power being drawn at a given time by each of the data center’s resource collections. The stress level is computed by one or several identical and redundant central authorities, and is communicated to each resource collection. Each collection balances the global stress against its average task load and delay sensitivity to determine how much power it should draw. The proposed control algorithm is only one of many strategies that can be used to reduce a data centers overall power consumption [16]. However, using an architecture like the one presented in this paper, i.e. one that supports autonomous resource operation and distributed control, emphasizes robustness and scalability.

The remainder of this paper is structured according to the following outline. In Section II, we define a data center model based on the previously described resource collections. In Section III, we present numerical results demonstrating the control strategy’s convergence rate and response to disturbances. In the Appendix, we provide a rigorous proof that under our model with justified assumptions, our distributed control strategy will converge to its unique fixed point. In the remaining Section IV, we introduce potential future work.

II. SYSTEM MODEL

A. Architecture

We define our system model in a general manner so that it may be applied similarly to multiple hierarchical levels. At the lowest level, we can define a resource collection as a single processor capable of frequency scaling and utilizing a variable number of cores to control its power consumption. In this manner, increased power translates to higher processing throughput, and from there to reduced time required to complete computational tasks. At a higher level, we can define a resource collection as a group of processors: for instance a single blade server or a larger group of such servers. In all such cases, we assume that the total system power consumption is dominated by the power used to perform computation, including the power used for temperature regulation. However, we acknowledge a base power consumption that cannot be completely eliminated, even at zero processing throughput. This is partially due to the central administrator. The administrator must be continually operational so it can respond to external stimuli such as an influx of new tasks, and determine when to return the collection’s power.

We broadly define all data center usage, either for computation or data retrieval/storage as tasks, which we consider as pre-assigned to a specific resource collection with appropriate computational capabilities. For example, highly parallel computations may be associated with a particular collection of graphics processing units, or GPUs, where a more sequential task may be assigned to a microprocessor. Thus each resource collection has a queue of tasks. The power that a collection is willing to draw increases with respect to the expected rate of tasks entering this queue. The exact relationship depends on the time-sensitivity of the tasks, and other real world parameters, for example: the priority status of the data center user requesting the task.

At any given time, the power consumption of every resource collection in the data center is available to one or more redundant and identical central administrators. These administrators compute a global power stress level used to regulate the power consumption of all resource collections. This is done by accessing a predetermined non-negative relative priority weight for each collection, and computing the weighted sum of all powers. This weighted sum is then input to a power stress function to compute the global power stress level. This function can be engineered to scale the overall power consumption as desired to conform to thermal requirements, or external factors like the market price the data center pays for electricity. Modification of this function also allows for simple implementation of demand response strategies like peak shifting, or general load shaping.

We acknowledge that such an architecture is not fully decentralized in its presented form. However, the responsibility of the central administrator is relatively light. It is also possible to shift to a more distributed paradigm to minimize the required networking and communication infrastructure. For instance, we could consider a mesh network that locally computes a power stress by allowing each mesh node to autonomously confer with neighboring resources. Regardless of how it is computed, the global power stress level is available at any time to every resource collection. The collections can asynchronously determine their individual power usage as a decreasing function of this one signal. In this manner, the average task delay of each individual resource collection balances against the overall power stress on the entire data center.

B. Power Control

Let \( p_i \) be the power drawn by resource collection \( i \), with \( i = 1, \ldots, n \). Given a particular power vector \( \vec{p} = (p_1, \ldots, p_n) \), the global power stress per unit (hereafter referred to as power stress) \( G \) is a function of \( \vec{p} \). \( G \) provides a measure of the system stress resulting from power consumption. Here we assume that

\[
G = g(w^T \cdot \vec{p})
\]  

(1)

with \( g(\cdot) \) an increasing and differentiable function of the weighted sum of the powers. The weights \( w_i > 0 \) may represent priorities assigned to each resource collection. A larger weight could indicate a collection that draws more power, or otherwise induces a disproportionate stress on the system. Each collection then induces a stress contribution of \( p_i G \) on the data center. We note that both the power stress function \( g(\cdot) \) and the weights \( w_i \)’s are determined and pre-set by the system administrator. They reflect how undesirable the current power configuration is to the administrator.

Each resource collection is associated with a delay function \( D_i(\cdot) \). For resource collection \( i \), if \( p_i \) power is drawn, then its delay \( D_i(p_i) \) is:

\[
D_i(p_i) := \begin{cases} 
  d_i(p_i), & p_i > p_i^* \\
  +\infty, & p_i \leq p_i^*,
\end{cases}
\]

(2)
where \( d_t : (\rho^*_i, +\infty) \rightarrow \mathbb{R}^{++} \) is assumed to be decreasing, strictly convex and twice differentiable, with \( \lim_{x \to +\infty} d_t(x) = +\infty \). \( \rho^*_i > 0 \) corresponds to the absolute minimum power that a resource collection must draw, below which the delay is infinite. As an example of such a delay function, consider the M/M/1 queue where the service rate provided by the resource collection is given \( r_i(p_i) \), an increasing function of the power consumption \( p_i \). In this case, if the jobs arrival rate is \( \rho_i \), then the average delay is \( \frac{1}{r_i(p_i) - \rho_i} \) (and infinite if \( r_i(p_i) \leq \rho_i \)), which can be checked to satisfy the above assumptions.

The central question concerning the system administrator is the trade-off between the power stress and the delay for the resource collections. For each resource collection \( i \), the trade-off between power and delay can be captured by the following objective:

\[
\lambda_i D_i(p_i) + p_i G = \lambda_i D_i(p_i) + p_i g(\bar{w}^T \cdot \bar{p}) \tag{3}
\]

where \( \lambda_i > 0 \) is the parameter that gives the relative trade-off between the two objectives: delay \( D_i(p_i) \) and power stress \( p_i G \). A large \( \lambda_i \) corresponds to a collection that is willing to induce a large stress in order to mitigate its average task delay. In contrast, a very small \( \lambda_i \) indicates that a collection would rather wait until the global power stress is lower before consuming the power required for computation. In each time step, each resource collection determines its locally optimal solution by minimizing the weighted objective given in Equation 3, as shown below:

\[
p' = \arg \min_q \lambda_i D_i(q) + qG \tag{4}
\]

The resource collection’s new power level, obtained by Equation 4, is then compared to its power level from the previous time step. If none of the resource collections in the datacenter alter their power consumption, then the power control scheme has necessarily converged to a fixed set of power consumptions and fixed global power stress level. A natural and intuitive distributed power update scheme is given in Algorithm 1, where each resource collection \( i \) chooses the power for the next iteration by minimizing the total objective. The current power stress per unit \( G^t \), common to all resource collections, is computed from the current power vector and made available to all resource collections.

### Algorithm 1 Distributed Power Update Scheme

1. Each resource collection starts with an arbitrary power \( p^0_i \)
2. \( t \leftarrow 0 \)
3. while not converged do
   4. \( G^t = g(\bar{w}^T \cdot \bar{p}^t) \)
   5. for \( i = 1, \ldots, n \) do
   6. \( p^{t+1} = h_i(G^t) = \arg \min_q \lambda_i D_i(q) + qG^t \)
   7. end for
   8. \( t \leftarrow t + 1 \)
4. end while

We can determine the easily verifiable conditions of the update system, which is characterized by \((h_1, \ldots, h_n, g)\) in Algorithm 1, under which the distributed power update scheme converges to an unique equilibrium, regardless of the initial power used by the collections. This feature is desirable since it not only implies that the system is stable, but also that the long-run performance of the system is characterized by the unique equilibrium: something that can be explicitly computed. To that end, we employ tools from fixed point theory. For full mathematical details, refer to the Appendix.

The required conditions for convergence are relatively light, and allow for a broad range of potential stress functions. Two simple examples—linear and exponential—are demonstrated in the following section.

The complexity of Algorithm 1 depends on the forms of the \( D_i \) and \( G \), and depending on these forms might be implemented as a quick computation, an optimization problem, or a simple look-up table.

### III. NUMERICAL RESULTS

While real-world datacenter workloads can be quite complex, and are not so easily modeled, we provide here an initial evaluation of our power control algorithm. Important potential future work includes implementation and evaluation against real-world workloads or standard benchmarks. For the following experiments, we use the normalized service rate function \( r_i(p) = 1 - e^{-p} \in [0, 1] \) and correspondingly normalize all task arrival rates to \( \rho_i \in [0, 1] \). We note that this service rate increases when more power is consumed, consistent with our model. Additionally, there is an asymptotic limit: a rate that cannot be exceeded for any power consumption. Such a limit models the thermal limitations of the processors comprising the resource collection. For all experiments, we considered a large number \((n = 100)\) number of randomly generated resource collections with normalized average arrival rate \( \rho_i \), stress/delay trade-off \( \lambda_i \), relative weight \( w_i \), and normalized initial power \( p^0_i \) all uniformly distributed on the unit interval \([0, 1]\) unless specifically noted otherwise.

#### A. Synchronous Convergence

We first assume that all resource collections update their power usage synchronously, and show a basic example of convergence. In Figures 1, 2, 3, and 4, we used the increasing stress function \( g(\bar{p}) = \bar{p} \), the direct linear function of the weighted sum of the powers.

We consider the system to have converged to its fixed point when no resource collection changes its power consumption by more than 1% in a single time step. In all presented plots demonstrating convergence, the system behavior is only plotted when no resource collection changes its power consumption by more than 1% in a single time step. In all presented plots demonstrating convergence, the system behavior is only plotted when no resource collection changes its power consumption by more than 1% in a single time step. In all presented plots demonstrating convergence, the system behavior is only plotted when no resource collection changes its power consumption by more than 1% in a single time step. In all presented plots demonstrating convergence, the system behavior is only plotted when no resource collection changes its power consumption by more than 1% in a single time step.

#### B. Asynchronous Convergence

More interesting and useful is what happens when the resource collections update asynchronously. In a real-world
Fig. 1. Convergence of power stress over time under our distributed control implementation. Each collection updates synchronously in every time step. We vary the relative delay sensitivity $\lambda_{max}$; the diversity (distribution width) in delay vs. power trade-off $\lambda \sim \text{uniform}(0, \lambda_{max})$. A larger $\lambda_{max}$ indicates a more diverse system.

system, it is difficult to guarantee that all of the distributed resources will update at the same time. These resources may operate on widely varying time scales, and their communication with the central administrator may not be entirely reliable. Additionally, verifying asynchronous convergence is a sufficient condition to guarantee synchronous convergence. In Fig. 2, we let each collection flip an independent Bernoulli coin at each time step to determine whether it updates. The bias of this coin corresponds to the frequency with which a particular collection updates. If the coin returns a deterministic true, the system reduces to the synchronous case. As we might expect, the system takes less time to converge when the collections are allowed to update more frequently, up to the limit of synchrony. However, all such update schemes converge to the same fixed point, regardless of update bias distribution.

Asynchronous convergence as demonstrated is an extremely useful property for our modeled data center. Relaxing synchrony allows each resource collection to update its power usage at its own discretion, such as between completing tasks. Asynchrony also increases system robustness by allowing for communication and update mishaps. Any collection may fail to update exactly when expected without affecting the general behavior of the overall system. This also removes any need for a global clock or update trigger signal.

C. System Expansion

Beyond having each collection update independently, we would also like to know how the system behaves when new resources are added to an already active system. In Fig. 3, we first allow the system to converge to its fixed point, then add a single new resource collection. Note that the power stress per collection decreases as new resources are added to the system, although the overall power stress naturally increases. As we might expect, adding a relatively large resource collection corresponds to a large system perturbation, thus requiring more time for the system to re-converge to the new fixed point.

Fig. 2. Convergence of power stress to equilibrium under our distributed control implementation. Unlike the synchronous case in Fig. 1, power levels are not updated every time step. Instead, resource collections update asynchronously. The chance a collection updates in a given time step is a fixed, random value drawn from the range indicated in the legend. The system behavior is simulated for several different update chance ranges. The relative power-delay sensitivity is held fixed at $\lambda = 1$, and the cost function is linear.

It is important to understand how the system reacts to such perturbations. Stable re-convergent behavior indicates that it is safe to expand the modeled data center without ceasing operation. Rather than shutting down the entire system and redesigning the control scheme, the distributed nature of this algorithm allows for relatively simple system augmentation.

Fig. 3. Synchronous re-convergence of power stress to equilibrium after a new resource collection is added to the system. System behavior is shown for several relative sizes of the added resource collection. The re-convergence time increases when larger collections are added to the system.

D. Convergence vs. System Parameters

We also explore how the system behaves with respect to its various parameters. In particular, Fig. 4 shows how long it takes our algorithm to converge based on the average task arrival rate and the delay to power trade-off. It also shows how the convergence time depends on the system diversity. In
general, more homogeneous systems converge more quickly. We note that the system's convergence behavior is relatively stable across a wide range of parameters. Accommodating a more diverse range of resources indicates a more generally useful algorithm.

E. Stress Function Design

Another concept we would like to consider is the effect of modifying the global stress function to control the system power equilibrium. By our definition of the global stress, the stress function will directly modify the stress itself. More power equilibrium. By our definition of the global stress, the of modifying the global stress function to control the system

![Graph](image1.png)

**Fig. 4.** System convergence time with respect to (1) average task arrival rate \( \rho \), (2) diversity of the average task arrival rate \( \rho_w \), where \( \rho \sim \text{uniform}(\frac{1}{2} - \rho_w, \frac{1}{2} + \rho_w) \), (3) delay sensitivity \( \lambda \), and (4) diversity of the delay sensitivity \( \lambda_w \), where \( \lambda \sim \text{uniform}(\frac{1}{2} - \lambda_w, \frac{1}{2} + \lambda_w) \).

stress functions can be directly implemented without concern for other users.

Future work may concern the implementation details of the distributed control algorithm discussed in this paper, including hardware implementation and testing. In particular, we would like to explore and verify the stability and convergence rates of the algorithm, as well as how it responds to disturbances of varying sizes.

Another potential consideration is physical proximity as a system parameter. Consider that in a real data center, a likely indicator of stress is the thermal state. Temperature and heat flow are intrinsically connected to location and packing density of resources. This also relates the problem back to the wireless communication context, in which stress is indicated by electromagnetic interference, which is a localized parameter.

APPENDIX: CONVERGENCE OF DISTRIBUTED POWER UPDATE SCHEME

A. Existence of Fixed Points

The update scheme in Algorithm 1 can be characterized by the following update function \( F : \mathbb{R}^n \rightarrow \mathbb{R}^n \), where \( \vec{p}^{t+1} = F(\vec{p}^t) \), with \( F(\vec{p}) = h \circ g(\vec{p}) = (h_1(g(u^T \cdot \vec{p})), \ldots, h_n(g(u^T \cdot \vec{p}))) \). We denote \( F^2 \) as the composition of \( F \) with itself: \( F^2 = F \circ F \). The next result characterizes the property of \( F \).

**Lemma 1:** \( F \) is order-reversing: \( \vec{p} > \vec{q} \Rightarrow F(\vec{p}) < F(\vec{q}) \).

**Remark 1:** Here \( \vec{p} > \vec{q} \) refers to \( \vec{p} \geq \vec{q} \) and at least one component of \( \vec{p} \) is larger than the corresponding component of \( \vec{q} \).

**Proof:** \( \vec{p} \geq \vec{q} \Rightarrow g(\vec{p}) > g(\vec{q}) \). Therefore, it suffices to show each \( h_i(\cdot) \) is a decreasing function. This is immediate by setting the derivative of \( h_i \)'s minimization argument to 0 and noting that \( D_i(\cdot) \) is a increasing function since \( D_i(\cdot) \) is strictly convex.

We first introduce the necessary terminology.

**Definition 1:** Let \( (P, \leq) \) be a partially ordered set and \( H : P \rightarrow P \) be a map.
1) A subset $S \subset P$ is called a chain if $S$ is totally ordered under $\leq$.
2) $H$ is called a poset-mapping continuous map if for each countable chain $\{c_i\}$ having a supremum, $H(\sup\{c_i\}) = \sup\{H(c_i)\}$.

The following theorem in [17] enables us to establish the existence of fixed points in unbounded posets (i.e. $\mathbb{R}^n_+$).

**Theorem 1:** Let $(P, \leq)$ be a partially ordered set and $H : P \to P$ be a poset-mapping continuous map. Assume the following three conditions hold:

1) $\exists p_1 \in P, p_1 \leq H(p_1)$;
2) every bounded countable chain in $\{x \in P \mid x \geq p_1\}$ has a supremum;
3) $\exists p_2 \in P, p_2 \geq p_1, p_2 \geq H(p_2)$.

Then $H$ has a fixed point.

**Lemma 2:** $F^2$ has a fixed point.

**Proof:** Let $H = F^2$ and it is monotonically increasing by Lemma 1. Since in addition $H$ is continuous (by the smoothness properties of $g$ and $D_i$), $H$ is poset-mapping continuous. This is easily seen by noting that:

- We have $H(\sup\{c_i\}) \geq H(c_i), \forall i$ by monotonicity, hence $H(\sup\{c_i\}) \geq \sup\{H(c_i)\}$.
- By continuity, $H(\sup\{c_i\}) \leq \sup\{H(c_i)\}$.

Further, note that $0 \leq F^2(0)$. In addition, since $F(\tilde{p}) > \bar{p}$, since more than $p_i$, power must be drawn to achieve finite objective value. Therefore, $F^2(\tilde{p}) < F(\tilde{p})$, $\forall \tilde{p}$. Take $\tilde{q} = F(\tilde{p})$, we then have $F^2(\tilde{q}) < \tilde{q}$, with $\tilde{q} \geq 0$. Hence, conditions 1 and 3 of Theorem 1 are satisfied. Condition 2 is again satisfied by the topological property of $\mathbb{R}^n_+$. Therefore, $F^2$ has a fixed point.

Next, we note that if $F^2$ has a unique fixed point, then repeated iterations of $F^2$ will always lead to convergence to that fixed point.

**Lemma 3:** If $F^2$ has a unique fixed point, then $F^{2t}(\tilde{p}^0)$ converges to the unique fixed point for any $\tilde{p}^0 \in \mathbb{R}^n_+$, as $t \to \infty$.

**Proof:** Take any $\tilde{p}^0 \in \mathbb{R}^n_+$. By the previous proof, we have $F^2(\tilde{p}) < F(\tilde{p}), \forall \tilde{p}$. Pick $\tilde{q}$ large enough such that $\tilde{q} \geq F(\tilde{p}), \tilde{q} \geq \tilde{p}^0$. Consequently, we have

- $0 \leq \tilde{p}^0 \leq \tilde{q}$.
- $F^2(\tilde{q}) < F(\tilde{p}) \leq \tilde{q}$.

Hence $F^{2t}(0) \leq F^{2t}(\tilde{p}^0) \leq F^{2t}(\tilde{q}), \forall t$. Note that $F^{2t}(0)$ converges to the unique fixed point since the sequence is bounded above by $\tilde{q}$ and monotonically increasing. Similarly $F^{2t}(\tilde{q})$ converges to the unique fixed point since the sequence is bounded below by $0$ and monotonically decreasing. Therefore, $F^{2t}(\tilde{p}^0)$ converges to the same unique fixed point.

**B. Uniqueness of Fixed Point**

Here we characterize the necessary and sufficient conditions under which $F^2$ has a unique fixed point. To that end, we introduce the induced scalar update function that has the equivalence of uniqueness of fixed point.

**Definition 2:** Let $\hat{p}^t = \hat{w}^T \cdot \hat{p}^t$. Define the scalar system update function $\hat{F} : \mathbb{R} \to \mathbb{R}$ as follows,

$$\hat{p}^{t+1} = \hat{F}(\hat{p}^t) = \sum_{i=1}^{n} w_i \cdot h_i(g(\hat{p}^t)).$$

(5)

We also denote $\hat{F}$ similarly to $F^2$: i.e. $\hat{F} = \hat{F} \circ \hat{F}$.

**Remark 2:** Observe that $\hat{F}$ is a decreasing function. Take $\hat{p} > \hat{q}$. Pick $\hat{p} > \hat{q}$ such that $\hat{w}^T \cdot \hat{p} = \hat{p}, \hat{w}^T \cdot \hat{q} = \hat{q}$. By Lemma 1, $F(\hat{p}) < F(\hat{q})$, leading to $\hat{F}(\hat{p}) = \hat{w}^T \cdot F(\hat{p}) < \hat{w}^T \cdot F(\hat{q}) = \hat{F}(\hat{q})$.

**Lemma 4:** $F^2$ has at most one fixed point if and only if $F^2$ has at most one fixed point.

**Proof:**

- First we show that if $F^2$ has at most one fixed point, then $F^2$ has at most one fixed point.

Assume that $F^2$ has two distinct fixed points $\hat{p}, \hat{q}$, with $\hat{p} > \hat{q}$. Then pick $\tilde{p} = h \circ g(\hat{w}^T \cdot h \circ g(\hat{p}))$. Note that $\hat{w}^T \cdot \tilde{p} = \tilde{p}$, since $\hat{p}$ is a fixed point of $F^2$. This then implies that $\hat{p}$ is a fixed point of $F^2$, since

$$F^2(\hat{p}) = h \circ g(\hat{w}^T \cdot h \circ g(\hat{p})) = h \circ g(\hat{w}^T \cdot h \circ g(\hat{p})) = \hat{p}$$

Similarly, pick $\tilde{q} = h \circ g(\hat{w}^T \cdot h \circ g(\hat{q}))$, then $\tilde{p} \neq \tilde{q}$ and $\tilde{q}$ is a fixed point of $F^2$, thereby resulting a contradiction.

- Next we show the converse: if $\hat{F}^2$ has at most one fixed point, then $F^2$ has at most one fixed point. Assume for contradiction purposes that $F^2$ has at least two fixed points: $F^2(\hat{p}) = \hat{p}, F^2(\hat{q}) = \hat{q}, \hat{p} \neq \hat{q}$. Then this implies that $\hat{w}^T \cdot \hat{p} \neq \hat{w}^T \cdot \hat{q}$, for otherwise $F^2(\hat{p}) = h \circ g(\hat{w}^T \cdot \hat{p}) = h \circ g(\hat{w}^T \cdot \hat{q}) = F^2(\hat{q})$, leading to $\hat{F}^2(\hat{p}) = \hat{F}^2(\hat{q})$, a contradiction.

Therefore, without loss of generality, assume $\hat{w}^T \cdot \hat{p} > \hat{w}^T \cdot \hat{q}$. However, $\hat{w}^T \cdot \hat{p}$ and $\hat{w}^T \cdot \hat{q}$ are both fixed points of $\hat{F}^2$. To see this, note that $\tilde{p} = F^2(\hat{p}) = h \circ g(\hat{F}(\hat{w}^T \hat{p}))$. Therefore, $\tilde{w}^T \cdot \tilde{p} = \hat{w}^T \cdot \hat{p} \circ g(\hat{F}(\hat{w}^T \hat{p})) = \hat{F}(\hat{w}^T \hat{p})$. Similarly, $\tilde{w}^T \cdot \tilde{q} = F^2(\hat{w}^T \hat{q})$. However, this contradicts the fact that $\hat{F}^2$ has a unique fixed point.

Note that $\hat{F}^2$ has a fixed point since $F^2$ has a fixed point by Lemma 2 and any fixed point $\hat{p}$ of $F^2$ induces a fixed point $\hat{w}^T \cdot \hat{p}$ of $F^2$. Therefore, Lemma 4 implies that $F^2$ has a unique fixed point if and only if $\hat{F}^2$ has a unique fixed point. In addition, $F$ is a decreasing function by Remark 2 and hence has a inverse function $F^{-1}$. This then allows us to define an easily-checkable condition under which $F^2$ has a fixed point.

**Definition 3:** The system in Algorithm 1 that is defined by $(h_1, \ldots, h_n, g)$ is called admissible if $\hat{F}(x) = F^{-1}(x), x > 0$ has a unique solution.
Remark 3: Since both $\hat{F}(x)$ and $\hat{F}^{-1}(x)$ are 1D functions, this is easily checked with negligible computational burden. In practice, the delay functions and the power stress functions are typically learned from past data and hence are usually represented in tabular forms. In such cases, one can plot $\hat{F}(x)$ and $\hat{F}^{-1}(x)$ to identify the number of intersections. This process is made easier if $\{h_1, \ldots, h_n, g\}$ have analytical forms, although in such cases, one can make direct analytical verifications. See examples in the next section and Fig. 6 where this condition is verified to hold.

C. Demonstration of Fixed Point Uniqueness

We first provide an illustration of the assertion from Remark 3: namely, that the update function intersects with its inverse exactly once. Fig. 6 shows an example of the described behavior for both a linear and a quadratic stress function.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig6.png}
\caption{Uniqueness of the scalar fixed point.}
\end{figure}

D. Convergence of Update Scheme

Finally, we state the main result concerning the convergence of the distributed power update scheme in Algorithm 1.

Theorem 2: Let $\{\hat{p}^t\}_{t=0}^{\infty}$ be a sequence of power vector iterates generated in Algorithm 1. If the update system in Algorithm 1 is admissible, then $F$ has a unique fixed point and $\hat{p}^* \in \mathbb{R}_n^+$ converges to the unique fixed point for any initial power vector $\hat{p}^0 \in \mathbb{R}_n^+$.

Proof: Since the update system in Algorithm 1 is admissible, $F^2$ has a unique fixed point, therefore implying that $F^2$ has a unique fixed point by Lemma 2 and Lemma 4. Further, by Lemma 3, $\hat{p}^{2t}$ converges to the unique fixed point for any initial power vector $\hat{p}^0 \in \mathbb{R}_n^+$. Similarly, $\hat{p}^{2t+1}$ converges to the unique fixed point $\hat{p}^*$ for any initial power vector $\hat{p}^0 \in \mathbb{R}_n^+$. Therefore, $\hat{p}^t$ converges to the unique fixed point for any initial power vector $\hat{p}^0 \in \mathbb{R}_n^+$.

$\hat{p}^*$ must necessarily be a fixed point of $F$. Since $F^2$ has a unique fixed point, $\hat{p}^*$ must be $F$'s unique fixed point.

ACKNOWLEDGMENT

Daniel Miller, Neal Master, and Zhengyuan Zhou are supported by Stanford University through the Stanford Graduate Fellowship (SGF) in Science & Engineering.

REFERENCES