A Hierarchical Demand Response Framework for Data Center

Power Cost Optimization Under Real-World Electricity Pricing

Cheng Wang, Bhuvan Urgaonkar, Qian Wang†, George Kesidis‡

Depts. of CSE, MNE†, and EE‡
The Pennsylvania State University
Technical Report CSE #13-010

Abstract

We study the problem of optimizing data center electric utility bill under uncertainty in workloads and real-world pricing schemes. Our focus is on using control knobs that modulate the power consumption of IT equipment. To overcome the difficulty of casting/updating such control problems and the computational intractability they suffer from in general, we propose and evaluate a hierarchical optimization framework wherein an upper layer uses (i) temporal aggregation to restrict the number of decision instants during a billing cycle to computationally feasible values, and (ii) spatial (i.e., control knob) aggregation whereby it models the large and diverse set of power control knobs with two abstract knobs labeled demand dropping and demand delaying. These abstract knobs operate upon a fluid power demand. The key insight underlying our modeling is that the power modulation effects of most IT control knobs can be succinctly captured as dropping and/or delaying a portion of the power demand. These decisions are passed onto a lower layer that leverages existing research to translate them into control decisions for real IT knobs. We develop a suite of algorithms for our upper layer that deal with different forms of input uncertainty (both stochastic and completely adversarial). An experimental evaluation of the proposed approach offers promising results: e.g., it offers net cost savings of about 25% and 18% to a streaming media server and a MapReduce-based batch workload, respectively.

I. INTRODUCTION

Data centers are large and growing consumers of power, in part due to their rapidly growing numbers and sizes to support various forms of cloud and utility computing. The global power
demand of data centers grew from 24GW in 2011 to 38GW in 2012 - a growth of 63% within a year [37]. Correspondingly, the power consumption of data centers is an important and growing contributor to their overall costs, with large data centers spending millions of dollars per year on their electric utility bills [1], [16]. 1 Optimizing the utility bill (by which we mean trading it off judiciously against performance/revenue) is, therefore, widely recognized as an important problem by cloud providers and other operators of data centers.

The Problem: A large body of work now exists in the broad area of power management which data centers can (and should) leverage for optimizing their utility bills. Our focus is on a subset of this work that carries out demand response (DR) by modulating the operation of the data center’s IT equipment. Examples of control “knobs” used for such DR include Dynamic Voltage Frequency Scaling (DVFS) for CPUs, entire server/cluster shutdown, load balancing and scheduling of jobs/requests, partial execution to consume lower power and offer low-quality results, just to name a few. 2 Despite this work, optimizing the utility bill using DR based on these IT knobs remains challenging due to:

- Complexity of IT Knobs: Most IT knobs offer discrete control options (e.g., DVFS states of a CPU, number of servers to shutdown, particular requests to turn away, etc.) which leads to combinatorial optimization formulations. Their operation is often intimately tied to idiosyncrasies of the software workload (e.g., load balancing or admission control at the unit of a “request,” scheduling at the unit of a “job” or a “process,” etc.) Finally, they exhibit significant diversity in the temporal (ranging from seconds to hours) and resource granularity (from single thread/CPU to much larger collections of resources) over which they operate. Due to these factors, even though these knobs are very well-understood individually, employing them collectively for optimizing a data center’s utility bill remains difficult.

- Complexity of Utility Pricing: Further contributing to the scalability challenge is the fact that utility bills are computed over significantly long periods of time (typically a month) compared

---

1Data centers also incur high costs towards building/maintaining the infrastructure for reliably delivering power to and cooling their equipment. These are one time or occasional investments unlike the recurring utility bills. Whereas our direct focus is on the utility bill, our work is also likely to be of indirect value in reducing the construction time costs (e.g., by allowing a data center to reduce its peak power consumption and hence provision a smaller-sized delivery infrastructure).

2Other forms of DR based on non-IT equipment (e.g., smart cooling systems) may also form part of the data center’s repertoire. Similarly, numerous techniques for cost optimization have also been explored on the “supply side” such as those based on employing energy storage, distributed generation (including renewables), participation in such programs as ancillary services, etc. We consider these complementary to our focus and outside the scope of the current work.
to the time granularity at which most IT knobs operate (seconds or minutes). Furthermore, real-world utility bills for large consumers such as data centers are not merely based on raw energy consumption over the billing cycle, but involve one (or a combination) of the following: (i) peak-based pricing, wherein a component of the electricity bill is dependent on the peak power drawn over the billing cycle [8] and (ii) time-varying pricing, wherein the per unit energy price fluctuates over time (there are many examples of this operating at different time granularity such as very fine time-scale spot prices [30], [33] or higher prices during periods of coincident peaks experienced by the utility [7]).

These factors can combine to result in optimization problem formulations that are (a) computationally intractable 3, and (b) difficult to cast and update upon changes within the workloads, pricing, or IT infrastructure. A significant portion of related work is concerned with optimization problems defined over much shorter time-spans using a small and specific set of knobs, and ignores/simplifies these complexities of pricing, allowing such work to overcome these scalability problems. A small set of papers do look at the utility bill optimization problem but tend to be focus on specific knobs and utility pricing schemes. We discuss these works in Section II.

**Research Contributions:** Our goal is to devise an optimization framework that can overcome these scalability problems while still being general enough to accommodate both (a) the multitude of IT knobs on which extensive research has been carried out individually, and (b) the idiosyncrasies of real-world utility pricing schemes. We would also like our approach to be capable of handling uncertainty in workloads and electricity prices. Towards this, we make three contributions.

- **Section II:** We propose a hierarchical framework that employs temporal aggregation and spatial (control knob) abstraction to achieve scalability and ease of casting/updating upon changes. The central idea underlying our modeling is our use of the abstract knobs of power demand dropping and delaying to encompass the many IT knobs that have been extensively studied (mostly individually). A second novelty over related work is our incorporation of various features of real world electricity pricing schemes into a single unified formulation. A final benefit of our hierarchical framework is the ease of plugging future power control techniques into the framework.

---

3Exceptions do exist and in some cases certain special structural properties allow for scalable optimization formulations - see Section II. We are, however, interested in the general case.
due to its modular design and the separation of the abstract knobs and the underlying real knobs.

- **Sections III and IV**: We develop a suite of DR algorithms capable of accommodating different kinds of input uncertainty (both stochastic and completely adversarial). (i) Our first algorithm is based on a stochastic dynamic program (SDP) that is somewhat unconventional due to its incorporation of both average and maximum in its objective (to capture peak-based pricing). (ii) When the data center workload is not amenable to simple stochastic modeling, the resulting SDP becomes computationally difficult to solve [2]. For such a case, we propose a “generalized stochastically bounded burstiness” (gsBB)-based [43] control approach which uses a general queuing model for the workload involving only a few statistical parameters. To our knowledge, this is the first attempt to use such a model for utility bill optimization in data centers (or even for other kinds of utility customers). (iii) Finally, we also develop a fully online algorithm (no prediction or historic data) for the case of a peak-based pricing scheme, and prove a competitive ratio of 2 or better.

- **Section V**: Finally, we study the efficacy of our framework through empirical case studies that use real-world traces for a variety of workloads (MPEG-based media server, MapReduce-based batch workload, a compound workload with both interactive and batch jobs) and utility pricing (derived from Duke electric [8]). Our initial findings are promising: e.g., our approach offers net cost savings of about 25% and 18% to our streaming media server and MapReduce-based batch workloads, respectively. Additionally, our experiments help us understand several subtelties of when and why DR is (or is not) useful for cost-efficacy.

**II. BACKGROUND AND OVERVIEW**

**A. Overview of Our Approach**

We propose a hierarchical optimization framework wherein the “upper” layer (the focus of this paper) carries out decision-making based on two complementary forms of aggregation. First, it employs temporal aggregation whereby it restricts the number of decision instances over the billing cycle to computationally feasible values. Second, it employs spatial (i.e., control knob) aggregation whereby it replaces the large and diverse set of IT knobs available to the data center with two abstract DR knobs labeled demand dropping and demand delaying that operate upon a fluid power
demand. Figure 1 presents an illustration of the main ideas underlying our approach.

The key insight underlying our modeling is that despite their immense diversity, all IT knobs for DR can be viewed as having the effect of either reducing (i.e., dropping) or postponing (i.e., delaying) a portion of the power consumption (or a combination of these effects). To appreciate this, consider the following concrete examples:

- **Demand Dropping**: A Web search application employing the knob of partial execution for some of its queries for meeting response time targets [40] is an example of a workload that is delay-sensitive and can drop some of its power needs (relative to that corresponding to the best quality results) to meet delay targets. Another example is a video server that exploits multiple fidelity videos that MPEG allows to guarantee fluent streaming [11] by sending lowering the video quality during overloads (with lower power consumption).

- **Demand Delaying**: For several batch workloads (e.g., a MapReduce-based set of tasks [41]), dropping is intolerable but delaying demand as long as it finishes before a deadline (typically with sufficient flexibility for scheduling) is acceptable.

- **Both Dropping and Delaying**: Some applications can have combinations of the above two. E.g., a search engine typically has a user-facing Web front-end as well as a backend component that does crawling and indexing, with these two components’ power demand modulated via dropping
and delaying, respectively.

Since many data center workloads exhibit uncertainty [28] as do electricity prices [22] we devise DR techniques that can adapt their behavior to workload and price evolution in an online manner. We devise three such control algorithms for sequential decision-making in Section IV. Our algorithms differ in the assumptions they make about input predictability (with correspondingly different efforts involved in “learning” their respective input models) and offer different computational complexities (i.e., scalability). A data center might find one of these more appropriate than others based on how this trade-off between input prediction accuracy and computational complexity applies to it.

Finally, the “bottom” layer of our hierarchical framework employs techniques from existing work to enforce the power modulation decisions made by the upper layer. To illustrate our approach, in Section V, we use existing work on power/performance trade-off associated with admission control and MPEG video quality modulation (for our streaming media server workload).

B. Related Work

We discuss three classes of related work.

Salient Power Management Knobs: A large body of literature focuses on exploiting IT control knobs to manage/reduce the “raw” power consumption of data centers. Examples of such work include the use of DVFS [3], [12], [14], DRPM [15], server shutdown [6], [21], VM consolidation [10], partial execution [40], job scheduling [42], load balancing [23], and combinations of these knobs [6], [27]. The efficacy of these IT control knobs has been explored individually by numerous research papers. However, these control knobs exhibit significant diversity in temporal granularity (e.g., DVFS at seconds, DRPM and server shutdown at minutes, etc) and control unit (e.g., load balancing at the unit of a “request,” consolidation at the unit of a VM, etc), which adds to the complexity of employing them collectively.

These are complementary to our work in that they provide salient ways for the lower level of our framework for translating our abstract control decisions.

Other Examples of Hierarchical Control: Power management in a hierarchical way is not new in the literature. Certain existing pieces of work have (either implicitly or explicitly) taken similar approaches (framework design), although not necessarily aiming at utility bill optimization. For
example, Raghavendra et al. proposes a hierarchical power capping framework for data centers which coordinates different power control techniques [27] so that they do not interfere with (or even cancel the effects of) one another. Kontorinis et al. presents a hierarchical architecture for distributed per-server UPSs to shave the power spikes of data centers [19]. Similar designs can also be found in [38], [39]. Common across all these works is the use of the notion of “power budgets/caps” that the higher levels determine and pass on to the lower levels that implement them.

Our abstraction is, in fact, a generalization of the power budgets used in these works because, in addition to specifying an upper bound on the power consumption during a slot (as in these works), we also specify how to achieve it via our combination of dropping and delaying.

*Utility Bill Optimization in Data Centers:* Most closely related to our work are some recent efforts that focus on the exact problem of utility bill optimization for the data center. More mature among these are papers that consider time-varying electricity pricing. For example, Urgaonkar et al. develop an Lyapunov optimization based technique to minimize operational cost using batteries [34]. Liu et al. propose distributed algorithms for geographical load balancing to exploit both the spatial and temporal aspects of time-varying pricing [22]. The time-varying pricing structure enables them to do DR within a single control window, or try to minimize the long-term expected costs wherein the complexity of the optimization problem can be reduced greatly with minimum optimality loss. On the other hand, peak-based electricity pricing, which is commonly seen in the real-world, has received less attention. The difficulty of DR under peak-based pricing is that the peak power consumption remains unknown until the end of the month, which results in scalability issues if DR is done at a fine time-scale. Some existing works avoid this difficulty by considering only a short planning period. For example, Liu et al. consider a hybrid of peak-based and time-varying pricing together with coincident peak charge and reduce power costs using robust optimization [22]. Xu et al. exploit the cost saving potential of partial execution for the web search requests under peak-based pricing [40]. However, both these works are only evaluated for a week-long horizon, and might not be scale to a month-long billing cycle. Furthermore, they focus on either a single IT knob or external energy source, whereas our goal is to accommodate a more general set of knobs.
III. Problem Formulation

In this section, we formulate the optimization problem that the upper layer of our hierarchical framework solves.

**Input Parameters:** We consider a slotted model wherein control decisions are made at the beginning of control windows of equal duration $\delta$. We denote by $T$ the number of such control windows within a single billing cycle, which constitutes our optimization horizon. Let the time-series $\{p_t : 1 \leq t \leq T\}$ denote the power demand of the data center over the optimization window of interest, with $0 \leq p_t \leq p_{\text{max}}$ denoting its power demand during the $t^{th}$ control window. Here $p_{\text{max}}$ denotes the maximum power demand the data center may pose during a single control window and would depend on the data center’s workload, its IT/cooling infrastructure, and the power delivery infrastructure provisioned by the data center. To represent the time-varying pricing employed in many electric tariffs (e.g., [30], [33]), we define a time-varying energy price of $\alpha_t$ dollars per unit energy (in units of $$/kWH). To represent peak-based charging (e.g., [8], [13], [40]), we define a peak price of $\beta$ dollars per unit peak power per billing cycle (in units of $$/kW). Based on evidence in existing work, we employ (i) a convex non-decreasing function to model the revenue loss due to demand dropping ($l_{\text{drop}}(\text{demand})$), and (ii) a linear non-decreasing function to model revenue loss (per slot of delay) due to demand delaying ($l_{\text{delay}}(\text{demand}) = k_{\text{delay}} \cdot \text{demand}$, $k_{\text{delay}} > 0$), where demand is in units of power [11], [23], [35].

**Decision Variables:** At the beginning of control window $t$ (or simply $t$ henceforth), the data center can have “residual” power demands due to delaying in the past (in addition to the newly incoming demand $p_t$). Let $r_t$ denote the aggregate residual demand at the beginning of time slot $t$. Denote as $y_{\text{max}}$ the peak demand admitted in any window during $[1, ..., T]$. The control actions to be taken by the data center during $t$ involve admitting, postponing, and dropping portions of $r_t$ and $p_t$. We denote as $d_t$ and $a_t$ the demand that is dropped and admitted out of $p_t$ respectively, $a'_t$ and $d'_t$ the demand that is dropped out of $r_t$. Finally, we denote as $\mathcal{A}$ the set $\{a_t\}_{t \in [1,T]}$, $\mathcal{A}'$ the set $\{a'_t\}_{t \in [1,T]}$, $\mathcal{D}$ the set $\{d_t\}_{t \in [1,t]}$, and as $\mathcal{D}'$ the set $\{d'_t\}_{t \in [1,t]}$ of decision variables. Table I summarizes our notation.

**Objective:** We choose as our data center’s objective the minimization of the sum of its utility bill and any revenue loss resulting from DR $\mathcal{O}(\{\mathcal{A}\}, \{\mathcal{D}\}, \{\mathcal{A}'\}, \{\mathcal{D}'\})$: $\beta y_{\text{max}} + \sum_t \left( \alpha_t(a_t + a'_t) + \right.$
$l_{\text{drop}}(d_t) + k_{\text{delay}}r_t$.

Notice that our objective does not have an explicit revenue term which might give the impression that it does not capture the data center’s incentive for admitting demand. It is important to observe that the incentive for admitting demand comes from the costs associated with dropping or delaying demand. Generally, one would have $l_{\text{drop}}(a) > \alpha_t(a) \cdot a$ to disallow scenarios where the data center prefers dropping all demand to admitting it. However, there can be situations (e.g., extremely high energy prices during coincident peaks [7], [22]) when this is not so. Finally, we could also include a revenue model of accepted demand, simply resulting in $\alpha_t(\cdot) < 0$ in our problem formulation.

**Constraints:** The aggregate demand in the data center at the beginning of $t$ is given by the new demand $p_t$, and any demand unmet so far (deferred from previous time slots) $r_t$. Since this demand must be treated via a combination of the following: (i) serve demand $a_t$ and $a'_t$, (ii) drop demand $d_t$ and $d'_t$, and (iii) postpone/delay demand $(r_{t+1})$ (to be served during $[t+1, \ldots, T]$), we have:

$$
(p_t - a_t - d_t) + (r_t - a'_t - d'_t) = r_{t+1}, \quad \forall t.
$$

Any residual demand or new demand that is not admitted or dropped during $t$ is postponed to the next time slot:

$$
r_t - a'_t - d'_t \geq 0, \quad \forall t.
$$

**TABLE I: Notation.**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>Total number of time slots in a billing cycle.</td>
</tr>
<tr>
<td>$\alpha_t$</td>
<td>Energy cost in time slot $t$ ($/kWh$).</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Peak power cost during the billing cycle ($/kW$).</td>
</tr>
<tr>
<td>$p_t$</td>
<td>Power demand during time slot $t$.</td>
</tr>
<tr>
<td>$p_{\text{max}}$</td>
<td>Maximum power demand allowed during a single time slot.</td>
</tr>
<tr>
<td>$y_{\text{max}}$</td>
<td>Maximum power demand allowed during a single time slot.</td>
</tr>
<tr>
<td>$r_t$</td>
<td>Aggregate residue power demand at the beginning of $t$.</td>
</tr>
<tr>
<td>$a_t$</td>
<td>Power demand admitted out of $p_t$ during time slot $t$.</td>
</tr>
<tr>
<td>$a'_t$</td>
<td>Power demand admitted out of $r_t$ during time slot $t$.</td>
</tr>
<tr>
<td>$d_t$</td>
<td>Power demand dropped out of $p_t$ during time slot $t$.</td>
</tr>
<tr>
<td>$d'_t$</td>
<td>Power demand dropped out of $r_t$ during time slot $t$.</td>
</tr>
<tr>
<td>$l_{\text{drop}}(d)$</td>
<td>Revenue loss due to dropping $d$ units of power demand, assumed convex and non-dec. in $d$.</td>
</tr>
<tr>
<td>$l_{\text{delay}}(d)$</td>
<td>Revenue loss due to delaying $d$ units of demand (per slot), assumed linear in $d$: $k_{\text{delay}}d$, $k_{\text{delay}} &gt; 0$.</td>
</tr>
</tbody>
</table>
\[ p_t - a_t - d_t \geq 0, \forall t. \]  

(3)

To keep our problem restricted to one billing cycle, we add an additional constraint that any delayed demand must be admitted by the end of our optimization horizon:

\[ r_{T+1} = 0. \]  

(4)

Alternate formulations that minimize costs over multiple billing cycles may relax the constraint above. The peak demand admitted \( y_{\text{max}} \) must satisfy the following:

\[ y_{\text{max}} \geq a_t + a'_t, \forall t. \]  

(5)

Finally, we have:

\[ y_{\text{max}}, a_t, d_t, a'_t, d'_t, r_t \geq 0, \forall t. \]  

(6)

**Offline Problem:** Based on the above, we formulate the following offline problem (called OFF):

Minimize \( O(\mathcal{A}, \mathcal{D}, \mathcal{A}', \mathcal{D}') \),

subject to \((1) - (6)\).

OFF will serve as a baseline against which we compare the efficacy of our stochastic/online algorithms in Section V. Owing to the following lemma, we can simplify our problem formulation by setting \( d'_t = 0, \forall t \):

**Lemma 1.** If \( l_{\text{delay}}(\cdot) > 0 \), then there always exists an optimal solution of OFF that never postpones some demand only to drop it in the future.

Our proof is based on a relatively straightforward contradiction-based argument. We omit it here for space and present it in the Appendix.

**Discussion:** We point out some limitations and possible generalizations of our formulations below:

- A key assumption in our modeling is what one might call “conservation” of power demand (as captured by constraint (1)). E.g., we do not capture possibilities such as delaying of some
power demand resulting in a change in overall power needs. Also, we assume that a power demand modulation decision does not affect the future external demand process. E.g., a request that is turned away (due to some power demand being “dropped”) does not return. Finally, we work with a $l_{\text{delay}}(\text{demand})$ function that is linear in the demand delayed per slot. A non-linear $l_{\text{delay}}(\text{demand}, \text{delay})$ would require that all the “residual” power demands that are postponed till a time slot and have not been admitted so far have to be recorded as state variables such that the delay costs can be evaluated based on their individual delay, which makes the framework unscalable. We make all of these simplifications for scalability (one of our primary goals), and leave it to future work to evaluate their impact on the efficacy of our approach.

- Our problem formulation is general enough to incorporate the so-called “coincident peak” as in [22]. This would be done by appropriately setting the $\alpha_t$ values for the windows when energy prices go up due to the occurrence of a coincident peak.

- An additional class of utility pricing schemes are load-dependent wherein higher energy prices are applied for higher power demands. An example of such a scheme is “tiered pricing” wherein some demand thresholds define different per unit prices for electricity; a special case has only one threshold, upon exceeding which a high rate (or additionally a “penalty”) is imposed [17], [26], [31]. We do not consider such pricing in this paper and leave algorithm design for such pricing to future work.

- Finally, one key benefit of our approach is the ease of casting (or re-casting upon changes in the environment) the overall optimization problem. In particular, the hierarchical nature of our approach implies that any changes only necessitate updates to the lower layer translations associated with these changes rather than the overall optimization framework. Section V implicitly illustrates this aspect of our approach.

IV. Decision-Making Under Uncertainty

A. Stochastic Dynamic Program

Since data center workloads can often be captured well via statistical modeling techniques [4], [22], we develop a stochastic dynamic programming (SDP) that leverages such modeling. A DR algorithm resulting from such an SDP is only partly online: it must be re-solved whenever the data
center updates its predictive models for workloads demands and/or electricity prices to yield the control policy to be used till the next time another adjustment of these models is done.

We choose as the goal of our SDP the minimization of the expectation of the sum of the data center’s electricity bill and demand modulation penalties over an optimization horizon, and denote it \( \mathcal{O}(\{A\}, \{A'\}, \{D\}) \) building upon the notation used in Section III. One key difficulty in our formulation arises due to the somewhat unconventional nature of our SDP wherein our objective is a hybrid of additive (total energy) and maximal (peak power) components.\(^4\) Consequently, we define as our state variable a combination of the residual demands \( r_t \) and the peak power admitted so far. Using \( y_t \) to denote the peak demand admitted during \([1, ..., t - 1]\), we represent the state at time \( t \) as the 2-tuple \( s_t = (r_t, y_t) \). The peak demand at the end of the optimization horizon \( y_{T+1} \) then corresponds to the variable \( y_{\text{max}} \) employed in OFF.

Since the power demands arriving in different control windows are only known stochastically, we introduce the notation \( P_t \) to denote these random variables, and use \( p_t \) to denote a particular realization of \( P_t \). Correspondingly, we denote by \( P_{[t]} = (P_1, ..., P_t) \) the history of the demand up to \( t \), and by \( p_{[t]} = (p_1, ..., p_t) \) its particular realization. We assume that the conditional probability distribution of the power demand \( P_t \), i.e., \( \Pr\{P_t = p_t | P_{[t-1]} = p_{[t-1]}\} \), is known. Similarly, we use \( \alpha_t \) to denote a particular realization of \( \Lambda_t \), the random variables for the time-varying prices in different control windows. we denote by \( \Lambda_{[t]} = (\Lambda_1, ..., \Lambda_t) \) and \( \alpha_{[t]} = (\alpha_1, ..., \alpha_t) \) the history of the price up to \( t \) and its particular realization, respectively. We assume the conditional probability distribution of time-varying price \( \Lambda_t \), i.e., \( \Pr\{\Lambda_t = \alpha_t | \Lambda_{[t-1]} = \alpha_{[t-1]}\} \), is known. We also assume that our control/modulation actions do not affect the demand/price arrival process.

Bellman’s optimality rules for our SDP can now be written as follows. For the last control window, we solve the following optimization problem (we denote it as SDP(T)) which gives \( V_T(s_T, p_{[T-1]}, \alpha_{[T-1]}) \):

\[
\min_{a_T,a'_T,d_T} \mathbb{E}\left\{ \beta y_{T+1} + \alpha_T(a_T + a'_T) + l_{\text{drop}}(d_T) + k_{\text{delay}} r_T | P_{[T-1]} = p_{[T-1]}, \Lambda_{[T-1]} = \alpha_{[T-1]} \right\},
\]

\(^4\)This represents an important difference from other existing SDP formulations in this area, such as [36], where their objective is optimizing a purely energy-based cost.
Subject to:

\[ p_T - a_T - d_T = 0; \quad r_T - a_T' = 0, \]
\[ y_{T+1} \geq y_T; \quad y_{T+1} \geq a_T + a_T', \]
\[ a_T, a_T', y_{T+1}, d_T \geq 0. \]

For control windows \( t = T - 1, \ldots, 1 \), we solve SDP(t) to obtain \( V_t(s_t, p_{[t-1]}, \alpha_{[t-1]}) \):

\[
\min_{a_t, a'_t, d_t} \mathbb{E} \left\{ \alpha_t(a_t + a'_t) + l_{\text{drop}}(d_t) + k_{\text{delay}}r_t + V_{t+1}(s_{t+1}, p_t) \mid P_{[t-1]} = p_{[t-1]}, \Lambda_{[t-1]} = \alpha_{[t-1]} \right\},
\]

Subject to:

\[
(p_t - a_t - d_t) + (r_t - a'_t) = r_{t+1},
\]
\[ r_t - a'_t \geq 0; \quad p_t - a_t - d_t \geq 0, \]
\[ y_{t+1} \geq y_t; \quad y_{t+1} \geq a_t + a'_t, \]
\[ a_t, a'_t, d_t, y_{t+1}, r_{t+1} \geq 0. \]

This SDP offers a runtime of \( O(R \cdot L_p^3 \cdot L_\alpha \cdot T) \) under the assumptions of stage-independence for demands and prices and denoting as \( O(R) \) the runtime of a sub-problem SDP(t) (\( L_p \) and \( L_\alpha \) are the number of discretization levels used for power demands and electricity prices, respectively). We conjecture that this SDP can be solved efficiently if the input prediction models are “simple” (e.g., stage-independent or first-order Markov chains). For more complex input models, however, the SDP is likely to run into the well-known “curses of dimensionality” and become intractable.

**Special Case of Only Dropping:** For some workloads, delaying is not tolerable and dropping is the only option. E.g., consider a Web search engine which is delay-sensitive but can do partial execution of search queries to save energy as long as response time guarantees are maintained [40]). For such cases, the state simplifies even further to \( s_t = y_t \) and the set of control variables shrinks to only \( d_t \) to describe dropping. This specialized SDP (that we label SDP_{Drop}) has a reduced runtime of \( O(R \cdot L_p^3 \cdot L_\alpha \cdot T) \). More interestingly, the structure of the optimal policy for SDP_{Drop} given by the following lemma (proof in the Appendix) provides a way of explicitly finding the optimal control policy [2]. Define \( \mu_t = \frac{a_t}{p_t}, \mu_t \in [0, 1], \mathcal{G}_t(\mu_t) = \mathbb{E} \{ \alpha_t(\mu_t p_t) \times \mu_t p_t + l_{\text{drop}}(p_t - \mu_t p_t) + V_{t+1}(\max\{\mu_t p_t, y_t\}) \} \) and \( \phi_t(y_t) = \arg\min_{\mu_t \in \mathcal{R}_+} \mathcal{G}_t(\mu_t). \)
Lemma 2. If the demands $p_t$ are independent across $t$, $SDP_{Drop}$ has the following threshold-based optimal control policy:

$$(a^*_t, d^*_t) = \begin{cases} 
(p_t - \phi_t p_t, \phi_t p_t), & \text{if } \phi_t \leq 1 \\
(p_t, 0), & \text{if } \phi_t > 1
\end{cases}$$

B. gSBB-based Control

When the data center workload is not stage-independent or is not even modeled well by a Markov process, the resulting SDP becomes computationally difficult to solve. For cases, we propose a gSBB-based control approach which uses a general queuing model for the workload involving only a few statistical parameters. We describe this next.

Let $\mu$ be the mean of the data center’s “raw” power demand. Here we consider more general demand modulation wherein units of demand will drop out and not return if they are deferred (or to be deferred) by more than $\tau^*$ time slots. In addition to $\mu$ and $\tau^*$, suppose the “raw” demand is also modeled by a “generalized stochastically bounded burstiness” (gSBB) curve \cite{43} $\{(\gamma, \phi(\gamma \tau^*))|\gamma > \mu\}$; i.e., a queue whose arrivals are the ”raw” demands and is served at rate $\gamma$ will have backlog $Q_\gamma$ such that

$$Pr(Q_\gamma \geq \gamma \tau^*) \leq \phi(\gamma \tau^*)$$

The basic procedure of this gSBB-based control is as follows (see Figure 2): The data center as a “server” in the gSBB queuing system runs at a maximum service rate $\gamma$, which is equivalent to regulating the peak power consumption to be within $\gamma$. Upon new arrival, if the queue is empty, the new power demand can be admitted directly. If the queue backlog is $\gamma \tau^*$, it would take the data center $\tau^*$ time slots to service all the backlog demand, and new demand will have to be dropped since it needs to wait for $\tau^*$ time slots has to be serviced. Otherwise, the new demand can enter the queue and be served within $\tau$ time slots in an FIFO order.

Note that the peak rate (and peak power consumption henceforth) of the queue-output is limited to $\gamma$. Under the peak-based pricing scheme, the objective of the top level demand modulation becomes one of selecting $\gamma$ to minimize the expected cost over a billing cycle of length $T$:

$$\min_{\gamma \geq 0} T \mu \int_0^\infty C(\tau)dF_\gamma(\tau) + \beta \gamma$$
where $T\mu$ is the expected total demand obtained from historical data. We define $C(\tau)$ as the “price/cost” (including energy cost, delay cost and dropping cost) imposed upon one unit of demand with delay $\tau$. For any unit of demand, if it is not delayed, the “price/cost” is equivalent to the energy cost $\alpha$; if it is delayed for $\tau$ time slots, the unit “price/cost” of $\alpha+k_{\text{delay}}\tau$ (or other forms of delay penalty) will be incurred for linear delay cost; if it is dropped, the “price/cost” will be $k_{\text{drop}}$. Here $C(\tau)$ can be viewed as an approximation for the actual unit “price/cost.” In Figure 3, we give examples of $C(\tau)$ with no delay, linear delay, and quadratic delay.

Correspondingly, we define $\Delta_{\gamma}$ as a random variable for the delay of unit demand given that the service rate of the departure process is $\gamma$, and denote as $F_{\gamma}(\tau)$ the CDF of $\Delta$, therefore

\[
F_{\gamma}(\tau) = \Pr(\Delta_{\gamma} \leq \tau) = \Pr\left(\frac{Q_{\gamma}}{\gamma} \leq \tau\right) = 1 - \phi(\gamma\tau)
\]

wherein the second equality holds since new demand will have delay $\frac{Q_{\gamma}}{\gamma}$ when the backlog is $Q_{\gamma}$. Then $dF_{\gamma}(\tau) = -d\phi(\gamma\tau)$. The objective cost function becomes:

\[
\min_{\gamma \geq 0} -T\mu \int_{0}^{\infty} C(\tau)d\phi(\gamma\tau) + \beta\gamma
\]

in which $C(\tau)$ and $\tau^*$ are pre-determined by the specific application running in the data center; $\phi(\gamma\tau^*)$ can be obtained from historical data by profiling $\phi(\gamma\tau^*)$.  

**Implementation Considerations:** To implement our gSBB-based control, the first step is to profile

---

5Note that the $\beta\gamma$ term approximates a peak power ($\gamma$) penalty. The $\gamma$ term here could be replaced instead by the approximation $(\mu + 2\sigma)(1 - \phi(\tau\gamma)) + 2\sigma$, i.e., mean power consumed plus two standard deviations (by an abuse of PASTA and “fluctuations increase delay,” the latter implying the standard deviation of the departures may be greater than than of the arrivals), where $\sigma$ is the standard deviation of the incident power demand process (arrivals).
\( \phi(\gamma \tau^*) \). Choose \( \gamma \) from \((0, p_{max}]\) and \( \tau^* \) from \([0, \tau_{max}]\) and obtain a table of \( \phi(\gamma, \tau^*) \) by repeatedly performing the basic procedure of gSBB-based control using given \( \gamma, \tau^* \) and historical data. Figure 4 shows an example of profiling \( \phi(\gamma \tau^*) \). The next step is to solve the above offline optimization problem. The solution \( (\gamma) \) of the problem together with \( \tau^* \) determined by the application can be used to perform gSBB-based control on the new demand. Note that the gSBB-based control can also be done in a partially online fashion by repeating the above steps with most recent historical data periodically.

**Discussion:** One important advantage of gSBB-based control over SDP is scalability. Recall that to keep the top level of our demand modulation framework scalable, OFF and SDP have to resort to a linear approximation of the delay cost. However, the gSBB-based formulation is able to incorporate any form of delay cost by defining \( C(\tau) \) appropriately while still being solved efficiently.

Another advantage is that the gSBB-based control does not assume any stochastic distribution/process for the demand. However, when the arrival process of the gSBB queue can be modeled as a diffusion process as in [29], \( \phi(\gamma \tau^*) \) can be approximated as an exponential function, which can be used to obtain a closed-form solution for the above optimization problem. We leave this to our future work.

**C. A Fully Online Algorithm**

We now present a fully online algorithm labeled ON\(_{Drop}\) and competitive analysis for it for the special case where only power demand dropping is allowed. Notice that the resulting problem setting is identical to that considered for SDP\(_{Drop}\) with the difference that whereas ON\(_{Drop}\) assumes adversarial power demand inputs, SDP\(_{Drop}\) was designed for power demands that can be described stochastically. The following lemma is key to the design of ON\(_{Drop}\):

**Lemma 3.** If only demand dropping is allowed, and \( l_{drop}(x) = k_{drop}x \), the optimal solution of OFF has a demand dropping threshold \( \theta \) of the following form: if we denote \( \hat{p}_t \) as the \( t^{th} \) largest demand
value in \( \{p_t\}_{t=1}^T \) and \( \hat{\alpha}_t \) the corresponding energy price, then \( \theta = \hat{p}_n \), where \( \beta - \sum_{t=1}^{n-1} (k_{\text{drop}} - \hat{\alpha}_t) \geq 0 \) and \( \beta - \sum_{t=1}^{n} (k_{\text{drop}} - \hat{\alpha}_t) \leq 0 \). As a special case, if \( \alpha_t = \alpha \) for all \( t \) (pure peak-based pricing),

\[
n = \left\lceil \frac{\beta}{k_{\text{drop}} - \alpha} \right\rceil.
\]

The proof of Lemma 3 implies that the optimal demand dropping threshold of OFF when only dropping is allowed can be found in an efficient way. If we can keep track of the \( \hat{p}_n \) in an online fashion, finally we can find the optimal demand dropping threshold after observing all the demand in the optimization horizon. So we exploit Lemma 3, to devise ON\( _{\text{Drop}} \) as follows:

**ON\(_{\text{Drop}}\): Online Algorithm With Only Dropping.**

1. **Initialization:** Set demand dropping threshold \( \theta_0 = 0 \).
2. **At time** \( t \), sort \( p_1, \ldots, p_t \) into \( \hat{p}_1 \geq \hat{p}_2 \geq \ldots \geq \hat{p}_t \).
3. **Update** \( \theta_t \) as follows: find index \( n \) such that \( \beta - \sum_{m=1}^{n-1} (k_{\text{drop}} - \hat{\alpha}_m) \geq 0 \) and \( \beta - \sum_{m=1}^{n} (k_{\text{drop}} - \hat{\alpha}_m) \leq 0 \); set \( \theta_t = \hat{p}_n \).
4. **Decision-making:** Admit \( \min(p_t, \theta_t) \), drop \( |p_t - \theta_t| + \).

For the special case of \( \alpha_t = \alpha \) for all \( t \) (pure peak-based pricing), since \( n = \left\lceil \frac{\beta}{k_{\text{drop}} - \alpha} \right\rceil \) according to Lemma 3, **step 3** can be simply becomes: if \( t < n \), \( \theta = 0 \); Otherwise, \( \theta = \hat{p}_n \).

**Theorem 1.** ON\(_{\text{Drop}}\) offers a competitive ratio of 2 under peak-based pricing.

**Corollary 1.** ON\(_{\text{Drop}}\) offers a competitive ratio of \( 2 - \frac{1}{n} \) under peak-based pricing when \( \alpha_t = \alpha \) for all \( t \).

We present our proofs in the Appendix. One useful way of understanding ON\(_{\text{Drop}}\) is to view it as a generalization of the classic ski-rental problem [18]: Increasing the dropping threshold is analogous to purchasing skis while admitting/dropping demands according to the existing threshold is analogous to renting them. The generalization lies repeating the basic ski-rental-like procedure after every \( n \) slots.

**Implementation Considerations:** At the **Initialization** step of ON\(_{\text{Drop}}\), the demand dropping threshold is set to 0, which will not be changed until \( t \geq n \). However, this is not practical in the real world (all the power demand before \( t \geq n \) is dropped) and results in the inefficiency of ON\(_{\text{Drop}}\). In fact, to reduce a monthly electricity bill, we can perform ON\(_{\text{Drop}}\) upon the most recent historical power demand data over a short period of time (e.g., one-day dataset) to obtain a better initial demand dropping threshold. We will evaluate this improved algorithm as ON\(_{\text{Drop}}^*\) in Section V.
V. Empirical Evaluation

In this section, we evaluate our approach using real-world workload power traces and utility prices. As described in Section II, since several prior works have looked at DR for time-varying utility prices, we restrict the focus of our evaluation to peak-based pricing (while noting that our framework is general enough to also capture time-varying prices including coincident pricing as in [22]).

A. Experimental Setup

Fig. 5: Workloads from the 14th day to the 16th day. The blue line is the original trace, and the red line is the time-of-day behavior of the trace.

Demand Traces: We employ three real-world demand traces (spanning 30 days) adopted from recent studies:

- **MediaServer**: a service request trace [20] from a live video streaming system which provides MPEG-4 video to its clients. For simplicity, we assume that the system will send a 10-min MPEG-4 video to incoming clients at one of 3000 kb/s or 1500 kb/s (MPEG-4 standard [24]) during a control window. We transform the MPEG-4 bit rate into power consumption based on data reported in [32] and generate the power demand trace for our MediaServer.
• **Facebook**: a power demand trace based on data reported from a Facebook data center [5] which runs MapReduce batch jobs.

• **Google**: a power demand trace based on data reported from a Google data center [9] which runs a mixture of MapReduce, Webmail, and Web search jobs.

Additionally, we create a **Synthetic** power demand series with an emphasis on including an unpredictable surge in power demand (e.g., as might occur due to a flash crowd), which is built by adding a high power surge to the demand for MediaServer on the 15th day. We show these power demand traces in Figure 5. Note that all of these real-world traces (except for Synthetic) exhibit a strong time-of-day behavior.

**Parameter Settings**: We set the energy price $\alpha = 0.046$/kWh and the peak power price $\beta = 17.75$/kW based on the Duke electric utility pricing [8]. For the time-varying pricing, we use the hourly electricity price for commercial or industrial customers ($> 2$MW) from 07/01/2012 to 07/30/2012 in zone A of National Grid U.S. [25] (see Figure 6). The energy price $\alpha_t$ ranges from 0.022$/kWh to 0.163$/kWh.

We work with a monthly electricity billing cycle, wherein the control window (one time slot) spans 10 minutes, so there are 4320 time slots in the optimization horizon. We set $k_{\text{delay}} = 0.02$/kWh based on [23], and $k_{\text{drop}} = \alpha + k_{\text{delay}}\tau^* = 0.246$, wherein $\tau^* = 10$ is the maximum delay allowed in the gSBB-based control. Our choice of $k_{\text{drop}}$ implies that it would be better to drop demand than to delay the demand for more than $\tau^*$ time slots. Because of the strong time-of-day behavior seen in the traces above, we use this time-of-day plus a zero-mean Gaussian noise in SDP and SDP$_{\text{Drop}}$. For ON$_{\text{Drop}}$, we train the algorithm using the demand of the first two days to obtain the initial dropping threshold, and then run the algorithm for the remaining 28-days.

**Our Evaluation Metric and Baseline**: Our baseline is the case with no demand modulation on the workload, thereby no revenue/performance loss attached to the power costs. The metric we use to evaluate our framework is **cost saving** which is the percentage reduction in costs due to a particular
algorithm compared to our baseline.

B. Cost Benefits

We show the cost compositions and savings for both real-world and synthetic traces in Figure 7. First, let us consider the MediaServer workload. During peak hours the streaming media system can either send low quality video or reject new clients both of which amount to power demand dropping. We find that both stochastic control and online control achieve near-optimal cost-savings (up to 25%) with revenue loss, with ON$_{Drop}$ providing a slightly higher cost-saving because the demand dropping threshold is trained using historical data. We also find that SDP$_{Drop}$ provides higher cost savings than ON$_{Drop}$ with more demand being dropped, which is because the benefits of dropping peak demand outweigh the costs incurred by dropping.

Next, let us look at the Facebook workload (Figure 7(b)), wherein dropping MapReduce batch jobs is not acceptable. However, the data center can delay demand via batch job scheduling or DVFS (or other techniques which have similar effects). We find that SDP provides 11.75% lower costs compared to OFF. At the same time, inappropriate demand delaying during non-peak time results in the inefficiency (and higher delay-related costs) of SDP.
Moving to the Google workload (Figure 7(c)), both of our abstract knobs are employed since the MapReduce jobs can be delayed and Web search engine can reduce power consumption via demand dropping (e.g., partial execution of search requests). We find that SDP provides near-optimal cost savings whereas gSBB provides little cost savings due to much larger demand being delayed. We also notice that neither gSBB nor SDP drop demand whereas a small portion of the “raw” demand is dropped in the optimal solution.

To see how our framework and algorithms work when there is unpredictable flash crowd, we evaluate the cost savings with Synthetic with both abstract knobs. In this case, SDP is not able to provide near-optimal cost saving whereas gSBB provides much better cost savings.

Finally, we observe that the total energy consumption in all cases are very similar before (i.e., with our baseline) and after DR. This means that the real peak over the billing cycle, which occurs rarely in the real-world, can be shaven by using our framework to greatly save the power costs without resulting in significant revenue loss (e.g., users might not come back once rejected (“raw” demand dropped)), which is true even when the only available knob is to drop (MediaServer). A second observation is that although OFF gives the optimal costs savings, it does not always have the least pure power costs, which is as expected since our goal is to optimize the total costs (including those due to revenue loss) instead of just the utility bill. This can also be explained by the (possibly) wrong decision makings (dropping more demand than the optimal solution during non-peak time) of other stochastic/online algorithms. We also find that sometimes the cost savings under the peak-based pricing might not always be significant even if the peak power costs take a large portion of the overall costs. For example, for Google, peak costs are almost 50% of the total costs, whereas the optimal cost saving is only 11.83%. MediaServer and Facebook have similar peak cost percentages, but much larger cost savings. This can be explained by the impact of workload properties, which we evaluate in later section.

Key insights: (i) Under peak-based pricing, our approach is able to provide significant cost savings (up to 25% for MediaServer) for real-world workloads without losing much “raw” demand. (ii) Our stochastic control and online control approaches achieve near-optimal cost-savings; however, they are not so beneficial when there is unexpected flash crowd (Synthetic), whereas the gSBB-based control is still able to handle workload unpredictability and provide cost-savings. (iii) In terms
of pure power costs, OFF does not always have the least costs since other control approaches might drop/delay more power demand (with higher performance/revenue loss) than the optimal solution and therefore incur lower pure power costs.

C. A Closer Look At Control Decisions

![Graphs showing control decisions for different demand dropping strategies.]

Fig. 8: An example of control decisions of real knobs corresponding to demand dropping.

We take a closer look at the actual IT knob control decisions (i.e., client-level admission control and MPEG-4 modulation) for the experiment involving our MediaServer workload trace. Our translation is based on the following simple heuristic: degrade the Quality-of-Service service (from 3000 kb/s to 1500 kb/s) to meet the demand dropping decision as much as possible; otherwise, reject some of the client requests to compensate for the target dropping power that is unmet. We show the real control decisions of our algorithms on the fifth day in Figure 8. We find that the control approaches of our framework make decisions similar to those made by OFF during the peak hour: all the clients are serviced via a lower bit rate; some clients are turned away. We also find that ON\textsubscript{Drop} drops more demand than ON\textsuperscript{*}\textsubscript{Drop} during the first 40 time slots from Figure 8(c)(d), which is in consistent with our expectation in Section IV that ON\textsuperscript{*}\textsubscript{Drop} provides better cost saving than ON\textsubscript{Drop} by choosing a higher initial demand dropping threshold.
D. Impact of Workload Properties

<table>
<thead>
<tr>
<th>Workload</th>
<th>Drop</th>
<th>Delay</th>
<th>Drop+Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>OFF</td>
<td>7.5</td>
<td>4.54</td>
</tr>
<tr>
<td>Facebook</td>
<td>18.21</td>
<td>18.15</td>
<td>15.15</td>
</tr>
<tr>
<td>MediaServer</td>
<td>24.59</td>
<td>23.4</td>
<td>22.85</td>
</tr>
<tr>
<td>Synthetic</td>
<td>45.01</td>
<td>43.78</td>
<td>42.96</td>
</tr>
</tbody>
</table>

TABLE II: Cost savings (%) offered by different algorithms under peak-based tariff.

In this section, we try to evaluate the impact of workload properties on cost savings under peak-based pricing. To help understand the features of our workloads, we select two workload properties that we intuitively expect to be informative about achievable cost savings: (i) the peak-to-average ratio (PAR), which captures the peak-shaving potential of the workload and (ii) peak width ($P_{70}$), which is defined as the percentage of the time slots in which the power demand value is larger than 70% of the peak power demand. We show these parameters in Fig. 5. Since we only focus on the shape of the power demand traces, we ignore the real-world implications of those traces and evaluate the cost-savings offered by different algorithms with different control on all the real-world traces as well as on the synthetic trace.

Let us first consider only using dropping. According to Lemma 3, the optimal demand dropping threshold under peak-based pricing is determined by $\hat{p}_n$, $n = \lceil (k_{\text{drop}} - \alpha) \rceil$ in the non-increasing array of $\{\hat{p}_t\}_{t=1}^T$. Since $n$ is fixed for all workloads given the pricing parameter settings, a larger $P_{70}$ implies higher probability of the optimal dropping threshold ($\theta$) being high, which means less power demand can be dropped to reduce the peak demand with lower cost savings. Among our workloads, Google has very “wide” peak/near-peak power values, which is the reason for the meager 7.52% cost savings. On the other hand, Synthetic exhibits “sharp” and “tall” peaks, for which a greater cost saving (45.01%) is possible.

Next, let us consider only using delaying. Deferring demand is possible only when a near-peak demand $p_t$ is followed immediately by much lower power demands, and the costs saving due to the resulting peak reduction is larger than the delay cost incurred. The lower and longer these succeeding low demand periods are (typically for larger PAR and smaller $P_{70}$), the better should be the cost savings achieved. Our experiment results verify these intuitions. As shown in Figure 5,
PAR increases drastically from Google, Facebook, MediaServer, Synthetic (in that order) while P_{70} decreases, and correspondingly, the cost savings improve (from 11.52% for Google to 42.47% for Synthetic).

Thirdly, let us consider the cost savings when both knobs are allowed: why are the cost savings very similar to those when only delaying is allowed? If a unit of delayed demand can be serviced with a small delay, the cost incurred is small compared to that for dropping. The low demand periods within short interval from near-peak demands in our workloads are plentiful, making dropping a rarely used knob. Only for Synthetic do we find that dropping demand helps improve cost saving by a non-trivial amount: from 42.47% with only delaying to 48.46% with both the knobs.

Finally, let us consider how our control algorithms perform. SDP and SDP_{Drop} works well under all workloads, since the workloads exhibit strong time-of-day behavior, implying our SDP has a reliable prediction model to work with. However, we observe that the cost saving from SDP for Synthetic becomes negative (-3.68%) when only delaying is employed, which suggests stochastic model is not suitable for capturing unexpected flash crowd. The gSBB-based control, although not very effective for Google and Facebook, provides near-optimal cost savings for MediaServer and even for Synthetic despite the unpredictable peak in the workload. ON_{Drop} performs well for almost all workload/control-knob combinations, except the 4.54% for Google with only dropping. Note that, in this case, there isn’t much room for savings to begin with (the optimal savings are only 7.52%). We report similar observations for ON^*_{Drop} with 6.28% cost savings for Google.

Key Insights: (i) For our workloads and parameters, delaying demand offers more benefits more than dropping, (ii) workload properties strongly affect the cost savings under peak-based tariff, and (iii) our stochastic control techniques are able to leverage workload prediction to offer near-optimal cost savings when there is no unexpected flash crowd.

E. Impact of demand modulation parameters

In this section, we evaluate the impact of demand modulation parameters, k_{delay} and k_{drop}, on the Google workload wherein both delaying and dropping can be employed. We vary the parameters by choosing k_{delay} from \{0.01, 0.02, 0.05\} and k_{drop} from \{0.146, 0.0246, 0.546\} \(k_{drop} = \alpha + k_{delay}\tau^*\). Besides cost saving, we also define another metric average delay as the total number of delay (in
time slots) for the modulated demand divided by number of time slots in the optimization horizon, which can roughly reflect the actual delay experienced by end users. We show the experiment results obtained by solving OFF in Figure 9.

As we expected, the average delay decreases as $k_{\text{delay}}$ increases (Figure 9(a)). On the one hand, we observe that when $k_{\text{delay}}$ is sufficiently small ($0.01$/kWh and $0.01$/kWh), the settings with $k_{\text{drop}} = 0.246$/kWh and $k_{\text{drop}} = 0.546$/kWh achieve similar cost savings, which implies that the total costs are not sensitive to $k_{\text{drop}}$. However, the same settings incur diverse average delay as in Figure 9(b)), which suggests different control decisions on the upper level. On the other hand, when $k_{\text{delay}}$ is sufficiently large ($0.05$/kWh), demand delaying becomes less useful than demand dropping based on the observation that larger $k_{\text{drop}}$ achieves significantly smaller cost saving but has similar average delay with the case of a smaller $k_{\text{drop}}$.

**Key insights:** (i) Different parameter settings might result in similar cost savings, but with diverse control decisions (thereby different user experience). (ii) When both abstract knobs are available, the decision of which knob the framework should be in favor of depends on the cost-benefit trade-offs between $k_{\text{delay}}$ and $k_{\text{drop}}$.

**VI. CONCLUSIONS AND FUTURE WORK**

We proposed a general framework for optimizing data center electric utility bills using the many control knobs that exist for modulating the power consumption of IT equipment. To overcome the difficulty of casting/updating such control problems and the computational intractability they are likely to suffer from in general, our optimization framework was hierarchical and employed both temporal aggregation and control knob abstraction. The key insight underlying our modeling was
that the power modulation effects of most IT control knobs could be thought of as dropping or delaying a portion of the power demand (or a combination of these). We developed a suite of algorithms for our upper layer (geared towards dealing with different forms of uncertainty, both stochastic and completely adversarial). Finally, we presented results from our empirical evaluation using real-world traces and workloads. Our initial findings were promising: e.g., our approach offered net cost savings of about 25% and 18% to a streaming media server and a MapReduce-based batch workload (both based on real-world traces and utility pricing), respectively. Additionally, our experiments helped us understand various subtleties of when and why DR was (or was not) useful for cost-efficacy.

REFERENCES


APPENDIX

A. Proof for Lemma 1

Proof: Let us assume an optimal solution $\mathcal{A}$ in which there exists some control window $t_1$ (where $1 \leq t_1 < T$) which violates the condition of our lemma. Therefore, of the demand $(r_{t,t+1} = p_t - a_{t,t} - d_{t,t})$ postponed (i.e., unmet) during $t_1$, there exists some portion $\nu$ (where $0 < \nu \leq a_{t,t}$), that is dropped during the control windows $[t_1+1, t_1+\tau]$. Let us focus on a portion $\nu_{t_2} > 0$ of $\nu$ that is dropped during the control window $t_2$ (where $t_1 < t_2 \leq t_1 + \tau$). Delaying $\nu_{t_2}$ over the period $(t_2 - t_1)$ and dropping it during $t_2$ contributes the following to the objective: $l_{\text{drop}}(\nu_{t_2}) + l_{\text{delay}}(\nu_{t_2})(t_2 - t_1)$. Let $C(\mathcal{A})$ denote the objective/cost offered by $\mathcal{A}$. 
Let us now compare $C(A)$ with the cost offered by an alternate solution $A'$ which drops $\nu_{t_2}$ during $t_1$ instead of delaying it. The two algorithms’ treatment of all other power demands (i.e., except for that for $\nu_{t_2}$) is exactly identical. This leads us to the following comparison of the different components of $C(A)$ and $C(A')$:

- **Energy costs**: Since both $A$ and $A'$ admit the same overall energy, they have identical energy costs.

- **Peak power cost**: $A'$ drops $\nu_{t_2}$ before $A$ does. Consequently, the peak power consumption of $A'$ cannot be worse (i.e., greater) than that of $A$.

- **Loss due to delaying or dropping demand**: Finally, whereas $A$ incurs a cost of $l_{\text{delay}}(\nu_{t_2})(t_2 - t_1) + l_{\text{drop}}(\nu_{t_2})$ for its treatment of $\nu_{t_2}$, $A'$ incurs a smaller cost of $l_{\text{drop}}(\nu_{t_2})$.

Combining the above, we find that $C(A') < C(A)$, which contradicts our assumption that $A$ was optimal.

**B. Proof for Lemma 2**

**Proof**: If the demands $p_t$ are independent across $t$, and we define $\mu_t = \frac{a_t}{p_t}, \mu_t \in [0, 1]$, SDP$_{\text{Drop}}$ becomes the following:

\[
V_T(y_T) = \min_{a_T, d_T} \mathbb{E}\{\alpha_T a_T + l_{\text{drop}}(d_T) + \beta y_{T+1}\} \\
= \min_{\mu_T \in [0, 1]} \mathbb{E}\{\alpha_T p_T \mu_T + l_{\text{drop}}(p_T(1 - \mu_T)) + \beta \max\{y_T, p_T \mu_T\}\}
\]

\[
V_t(y_t) = \min_{a_t, d_t} \mathbb{E}\{\alpha_t a_t + l_{\text{drop}}(d_t) + V_{t+1}(y_{t+1})\} \\
= \min_{\mu_t \in [0, 1]} \mathbb{E}\{\alpha_t p_t \mu_t + l_{\text{drop}}(p_t(1 - \mu_t)) + V_{t+1}(\max\{y_t, p_t \mu_t\})\}, \ t = 1, ..., T - 1
\]

Define $G_t(\mu_t) = \mathbb{E}\{\alpha_t p_t \mu_t + l_{\text{drop}}(p_t + V_{t+1}(\max\{y_t, p_t \mu_t\}))\}$, and suppose $G_t$ is convex, which will be proved later, and $G_t$ has an unconstrained minimum with respect to $y_t$, denoted by $\phi_t$: $\phi_t = \arg\min_{\mu_t \in \mathbb{R}^+} G_t(\mu_t)$. Then, in view of the constraint $0 \leq \mu_t \leq 1$ and the convexity of $G_t$, it is easily seen that an optimal policy can determined by the sequence of scalars $\{\phi_1, \phi_2, ..., \phi_T\}$ and has the form

\[
\mu^*_t(y_t) = \begin{cases} 
\phi_t, & \text{if } \phi_t \leq 1 \\
1, & \text{if } \phi_t > 1 
\end{cases}
\]
For SDP\textsubscript{Drop}, we have
\[(a^*_t, d^*_t) = \begin{cases} 
(\phi_t p_t, p_t - \phi_t p_t), & \text{if } \phi_t \leq 1 \\
(p_t, 0), & \text{if } \phi_t > 1 
\end{cases}\]

Now we will prove the convexity of the cost-to-go functions \(V_t\) (and hence \(G_t\)), so that the minimizing scalars \(\phi_t\) exist. We use induction to prove the convexity.

For the base case, as shown above, an optimal policy at time \(T\) is given by
\[\mu^*_T(y_T) = \begin{cases} 
\phi_T, & \text{if } \phi_T \leq 1 \\
1, & \text{if } \phi_T > 1 
\end{cases}\]

Furthermore, by plugging \(\mu^*_T\) back into \(V_T\), we have
\[V_T(y_T) = \begin{cases} 
\mathbb{E}\{\alpha_T p_T \phi_T + l_{\text{drop}}(p_T(1 - \phi_T)) \\
+ \beta \max\{y_T, p_T \phi_T\}\}, & \text{if } \phi_T \leq 1 \\
\mathbb{E}\{\alpha_T p_T + \beta \max\{y_T, p_T\}\}, & \text{if } \phi_T > 1 
\end{cases}\]

which is a convex function since \(\max\{y_T, \cdot\}\) is convex function of \(y_T\). This argument can be repeated to show that for all \(t = T - 1, \ldots, 1\), if \(V_{t+1}\) is convex, then we have
\[V_t(y_t) = \begin{cases} 
\mathbb{E}\{\alpha_t p_t \phi_t + l_{\text{drop}}(p_t(1 - \phi_t)) \\
+ V_{t+1}(\max\{y_t, p_t \phi_t\})\}, & \text{if } \phi_t \leq 1 \\
\mathbb{E}\{\alpha_T p_T + V_{t+1}(\max\{y_t, p_t \phi_t\})\}, & \text{if } \phi_t > 1 
\end{cases}\]

is also convex function of \(y_t\). By induction, \(V_t\) is convex for all \(t = 1, \ldots, T\). Thus, the optimality of the above policy is guaranteed. \[\blacksquare\]
C. Proof for Lemma 3

Proof: Without demand delaying, \( r_{i,t} = 0, i \in h(t) \forall t \). Therefore, we have \( d_t = p_t - x_t, \forall t \). The objective becomes:

\[
\begin{align*}
&\min_{a_t:y_{\max}} \sum_t \{\alpha_t a_t + k_{\text{drop}}(p_t - a_t)\} + \beta y_{\max} \\
= &\min_{a_t:y_{\max}} \sum_t (\alpha_t - k_{\text{drop}})a_t + \beta y_{\max}
\end{align*}
\]

Since \( y_{\max} \geq a_t, \forall t \), \( y_{\max} \) will be equal to the largest \( a_t \) in the optimal solution. For \( p_t \geq y_{\max} \), we have \( a_t = y_{\max} \), whereas for \( p_t < y_{\max} \), we have \( a_t = p_t \). We denote this as: \( a_t = p_t - (p_t - y_{\max})I_{\{p_t \geq y_{\max}\}} \), where \( I \) is the standard indicator function. The optimal value is:

\[
\begin{align*}
&\min_{y_{\max}} \sum_t (\alpha_t - k_{\text{drop}}) (p_t - (p_t - y_{\max})I_{\{p_t \geq y_{\max}\}}) + \beta y_{\max} \\
= &\min_{y_{\max}} \sum_t (k_{\text{drop}} - \alpha_t)(p_t - y_{\max})I_{\{p_t \geq y_{\max}\}} + \beta y_{\max}
\end{align*}
\]

We denote as \( V(y_{\max}) \) the expression \( \sum_t (k_{\text{drop}} - \alpha_t)(p_t - y_{\max})I_{\{p_t \geq y_{\max}\}} + \beta y_{\max} \), and prove that \( V(y_{\max}) \) is a convex piecewise-linear function of \( y_{\max} \). We sort the array \( \{p_t\} \) into \( \{\hat{p}_t\} \) such that \( \hat{p}_1 \geq \hat{p}_2 \geq \ldots \geq \hat{p}_{T'} \). We denote as \( \hat{\alpha}_t \) the energy pricing corresponding to \( \hat{p}_t \). Observing that multiple time slots might have the same power demand value, we denote \( p_i^{k_i}, 1 \leq i \leq T' \) (\( T' \) is the number of distinct power demand values in \( \{\hat{p}_t\} \)) as the \( i\text{th} \) largest value in \( \{\hat{p}_t\} \) with \( k_i \) time slots having the same value \( \hat{p}_i \). Then the following holds:

- If \( y_{\max} > p_1^{k_1} \), none of the power demand values are larger than \( y_{\max} \). This implies that \( V(y_{\max}) = \beta y_{\max} \).
- If \( p_1^{k_1} \geq y_{\max} > p_2^{k_2} \), \( k_1 \) power demand values are larger than \( y_{\max} \). This implies that \( V(y_{\max}) = [\beta - \sum_{t=1}^{k_1} (k_{\text{drop}} - \alpha_t)]y_{\max} + \sum_{t=1}^{k_1} (k_{\text{drop}} - \alpha_t)p_1^{k_1} \).
- \( \ldots \)
- Finally, if \( y_{\max} \leq p_T^{k_T} \), all power demand values are larger than \( y_{\max} \). This implies that \( V(y_{\max}) = (\beta - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t))y_{\max} + \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t)p_t \).

The slope of \( V(y_{\max}) \) does not increase as \( y_{\max} \) decreases since \( k_{\text{drop}} - \alpha_t > 0 \) for \( \forall t \in [1, \ldots, T] \). Therefore, \( V(y_{\max}) \) (and hence the objective function of OFF with only demand dropping) is a convex piecewise-linear function of \( y_{\max} \). If there exists an index \( n \) such that \( \beta - \sum_{t=1}^{n-1} (k_{\text{drop}} - \alpha_t) \geq \).
\[ 0 \text{ and } \beta - \sum_{t=1}^{n}(k_{\text{drop}} - \hat{\alpha}_t) \leq 0, \text{ the optimal demand dropping threshold } \theta \text{ of OFF will be } \hat{p}_n; \text{ otherwise, } \theta = 0. \text{ Furthermore, if } \alpha_t = \alpha, \, n = \left[ \frac{\beta}{(k_{\text{drop}} - \alpha)} \right], \text{ which can be verified easily. This optimal threshold can be found in } O(T \cdot \log T), \text{ the time needed for sorting the array } \{p_t\}_{t=1}^{T} \text{ plus the time for finding } n. \]

\[ \square \]

D. Proof for Theorem 1

**Proof:** From the algorithm description of ON\(_{\text{Drop}}\), we can obtain the following properties:

**Property 1:** The demand dropping threshold of ON\(_{\text{Drop}}\) keeps non-decreasing and is guaranteed to converge to the optimal threshold after all demands in the optimization horizon are observed.

**Property 2:** The demand dropping threshold of ON\(_{\text{Drop}}\) never exceeds the optimal threshold. Furthermore, if we denote as \(a^d_t\) the admitted demand by ON\(_{\text{Drop}}\) at time \(t\), and \(a^m_t\) the admitted demand by OFF at time \(t\) when only the demand values in the first \(m\) time slots are observed, \(1 \leq m \leq T\), then \(a^d_t \leq a^m_t\).

It is very easy to verify **Property 2** by induction. For **Property 1**, according to ON\(_{\text{Drop}}\), if \(p_t \leq \theta_{t-1}\) at time \(t\), it will affect neither \(\beta - \sum_{m=1}^{n-1}(k_{\text{drop}} - \hat{\alpha}_m)\) nor \(\beta - \sum_{m=1}^{n}(k_{\text{drop}} - \hat{\alpha}_m)\); therefore \(\theta_t = \theta_{t-1}\). If \(p_t > \theta_{t-1}\), assuming \(p_t\) has index \(w\) in \(\{\hat{p}_m\}_{m=1}^{t}, 1 \leq w < n + 1\), apparently \(\beta - \sum_{m=1}^{w-1}(k_{\text{drop}} - \hat{\alpha}_m) \geq 0\). If further we have \(\beta - \sum_{m=1}^{w}(k_{\text{drop}} - \hat{\alpha}_m) \leq 0\), then \(\theta_t = p_t > \theta_{t-1}\). Otherwise \(\theta_{t-1} \leq \theta_t < p_t\). So \(\theta_t\) keeps non-decreasing. The convergence of ON\(_{\text{Drop}}\) is guaranteed by the proof of Lemma 3. See Figure 10 for an illustration of the properties and how ON\(_{\text{Drop}}\) works. A direct corollary we can obtain from **Property 1** and **2** is that \(a^m_t \leq a^{m+1}_t, \forall t \in [1, ..., T]\)

\[ \text{Fig. 10: An illustration of ON}_{\text{Drop}}\]. \(p_t\) is the original power demand; \(\theta_t\) is the optimal demand dropping threshold when only the demand values in the first \(t\) time slots are observed; \(a^d_t\) is the admitted demand by ON\(_{\text{Drop}}\). \]
and $\forall m \in [1, ..., T - 1]$. According to Property 1 and 2, the total costs of $\text{OND}_\text{Drop}$ and OFF are:

$$\text{Cost}_{\text{OFF}} = \beta T + \sum_{t=1}^{T} \alpha_t a_t^T + k_{\text{drop}} \sum_{t=1}^{T} (p_t - a_t^T)$$

$$= \beta T + k_{\text{drop}} \sum_{t=1}^{T} p_t - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T$$

$$\text{Cost}_{\text{OND}_\text{Drop}} = \beta T + \sum_{t=1}^{T} \alpha_t a_t^d + k_{\text{drop}} \sum_{t=1}^{T} (p_t - a_t^d)$$

$$= \beta T + k_{\text{drop}} \sum_{t=1}^{T} p_t - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^d$$

We define competitive ratio CR of $\text{OND}_\text{Drop}$ as the upper bound of $\frac{\text{Cost}_{\text{OND}_\text{Drop}}}{\text{Cost}_{\text{OFF}}}$ under all possible workload scenarios, which is:

$$\text{CR} = \sup_{\{p_t\}_{t=1}^{T}} \frac{\text{Cost}_{\text{OND}_\text{Drop}}}{\text{Cost}_{\text{OFF}}}$$

Note that the only difference between the denominator and numerator is $\sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^d$ and $\sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T$. From Property 2 we know that $\sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T \leq \sum_{t=1}^{T} (k_{\text{drop}} - \bar{\alpha}_t) a_t^d$, so $\frac{\text{Cost}_{\text{OND}_\text{Drop}}}{\text{Cost}_{\text{OFF}}} \geq 1$. However, we can still find a bound $\sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T - \sum_{t=1}^{T} (k_{\text{drop}} - \bar{\alpha}_t) a_t^d \leq \beta (T - \theta_1)$ (when $\alpha_t = \alpha$, the upper bound reduces to $(k_{\text{drop}} - \alpha)(n - 1)\theta_T$, $n = \lceil \frac{\beta}{|k_{\text{drop}} - \alpha|} \rceil$) to make sure that CR will not go to infinity. We will prove this upper bound later. Now with this bound we have:

$$\frac{\text{Cost}_{\text{OND}_\text{Drop}}}{\text{Cost}_{\text{OFF}}} = \frac{\beta T + k_{\text{drop}} \sum_{t=1}^{T} p_t - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^d}{\beta T + k_{\text{drop}} \sum_{t=1}^{T} p_t - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T}$$

$$\leq \frac{\beta T + k_{\text{drop}} \sum_{t=1}^{T} p_t - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T + \beta (T - \theta_1)}{\beta T + k_{\text{drop}} \sum_{t=1}^{T} p_t - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T}$$

$$= 1 + \frac{\beta (T - \theta_1)}{\beta T + k_{\text{drop}} \sum_{t=1}^{T} p_t - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T}$$

$$= 1 + \frac{1 - \frac{\theta_1}{\beta T}}{1 + \frac{k_{\text{drop}} \sum_{t=1}^{T} p_t - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T}{\beta T}}$$

$$< 2$$

Specifically, when $\alpha_t = \alpha$, we have $\frac{\text{Cost}_{\text{OND}_\text{Drop}}}{\text{Cost}_{\text{OFF}}} \leq 2 - \frac{1}{n}$. 
Next we prove \( \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^d \leq \beta(\theta_T - \theta_1) \).

\[
\sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^d = \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t)(a_t^T - a_t^d)
\]

\[
= \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t)(a_t^2 - a_t^1) + \ldots + \sum_{t=1}^{T-1} (k_{\text{drop}} - \alpha_t)(a_t^T - a_t^{T-1})
\]

Since \( \theta_m \) (1 ≤ m ≤ T) equals to the optimal demand dropping threshold of OFF when only the first \( m \) time slots are considered, we have

\[
\sum_{t=1}^{m} (k_{\text{drop}} - \alpha_t)(a_t^{m+1} - a_t^m)
\]

\[
= \sum_{1 \leq t \leq m} (k_{\text{drop}} - \alpha_t)(\theta_{m+1} - \theta_m)
\]

\[
= (\theta_{m+1} - \theta_m) \sum_{1 \leq t \leq m} (k_{\text{drop}} - \alpha_t)
\]

If we only consider the first \( m \) time slots, according to Lemma 3, \( \beta \) should be larger than or equal to \( \sum_t (k_{\text{drop}} - \alpha_t) \), and \( t \) denotes the time slots during which \( p_t > a_t^m \). Therefore, \( (\theta_{m+1} - \theta_m) \sum_{1 \leq t \leq m} (k_{\text{drop}} - \alpha_t) \leq (\theta_{m+1} - \theta_m) \beta \).

Finally,

\[
\sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^T - \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t) a_t^d
\]

\[
= \sum_{t=1}^{T} (k_{\text{drop}} - \alpha_t)(a_t^2 - a_t^1) + \ldots + \sum_{t=1}^{T-1} (k_{\text{drop}} - \alpha_t)(a_t^T - a_t^{T-1})
\]

\[
\leq \beta(\theta_2 - \theta_1) + \ldots + \beta(\theta_T - \theta_{T-1})
\]

\[
= \beta(\theta_T - \theta_1)
\]

which completes our proof for the competitive ratio of ON\text{Drop}.

\[\square\]

**E. Proof for Corollary 1**

*Proof:* From the algorithm description of ON\text{Drop}, we can obtain the following properties:
**Property 1:** The demand dropping threshold of ON\textsubscript{Drop} keeps non-decreasing and is guaranteed to converge to the optimal threshold after all demands in the optimization horizon are observed.

**Property 2:** The demand dropping threshold of ON\textsubscript{Drop} never exceeds the optimal threshold. Furthermore, if we denote as \(a^d_t\) the admitted demand by ON\textsubscript{Drop} at time \(t\), and \(a^m_t\) the admitted demand by OFF at time \(t\) when only the demand values in the first \(m\) time slots are observed, \(1 \leq m \leq T\), then \(a^d_t \leq a^m_t\).

It is very easy to verify the above properties by the algorithm details of ON\textsubscript{Drop} and Lemma 3. See Figure 11 for an illustration of the properties and how ON\textsubscript{Drop} works. According to **Property 1** and 2, the total costs of ON\textsubscript{Drop} and OFF are:

\[
\text{Cost}_{\text{OFF}} = \beta T \theta + \alpha \sum_{t=1}^{T} a^T_t + k_{\text{drop}} \sum_{t=1}^{T} (p_t - a^T_t) \\
= \beta T \theta + k_{\text{drop}} \sum_{t=1}^{T} p_t - (k_{\text{drop}} - \alpha) \sum_{t=1}^{T} a^T_t
\]

\[
\text{Cost}_{\text{ONDrop}} = \beta T \theta + \alpha \sum_{t=1}^{T} a^d_t + k_{\text{drop}} \sum_{t=1}^{T} (p_t - a^d_t) \\
= \beta T \theta + k_{\text{drop}} \sum_{t=1}^{T} p_t - (k_{\text{drop}} - \alpha) \sum_{t=1}^{T} a^d_t
\]

We define competitive ratio CR of ON\textsubscript{Drop} as the upper bound of \(\frac{\text{Cost}_{\text{ONDrop}}}{\text{Cost}_{\text{OFF}}}\) under all possible workload scenarios, which is:

\[
\text{CR} = \sup_{\{p_t\}_{t=1}^{T}} \frac{\text{Cost}_{\text{ONDrop}}}{\text{Cost}_{\text{OFF}}}
\]
Note that the only difference between the denominator and numerator is $\sum_{t=1}^T a_t^d$ and $\sum_{t=1}^T a_t^T$.

From Property 2 we know that $\sum_{t=1}^T a_t^T \leq \sum_{t=1}^T a_t^d$, so $\frac{\text{Cost}_{\text{ONDrop}}}{\text{Cost}_{\text{OFF}}} \geq 1$. However, we can still find a bound $\sum_{t=1}^T a_t^T - \sum_{t=1}^T a_t^d \leq (n-1)\theta_T$, $n = \lceil \frac{\beta}{(k_{\text{drop}}-\alpha)} \rceil$ to make sure that CR will not go to infinity. We will prove this bound later. Now with this bound we have:

$$\frac{\text{Cost}_{\text{ONDrop}}}{\text{Cost}_{\text{OFF}}} = \frac{\beta \theta_T + k_{\text{drop}} \sum_{t=1}^T p_t - (k_{\text{drop}} - \alpha) \sum_{t=1}^T a_t^d}{\beta \theta_T + k_{\text{drop}} \sum_{t=1}^T p_t - (k_{\text{drop}} - \alpha) \sum_{t=1}^T a_t^T} \leq \frac{\beta \theta_T + k_{\text{drop}} \sum_{t=1}^T p_t - (k_{\text{drop}} - \alpha) (\sum_{t=1}^T a_t^T - (n-1)\theta_T)}{\beta \theta_T + k_{\text{drop}} \sum_{t=1}^T p_t - (k_{\text{drop}} - \alpha) \sum_{t=1}^T a_t^T}$$

(Since $n = \lceil \frac{\beta}{(k_{\text{drop}}-\alpha)} \rceil$)

$$= 1 + \frac{\beta \theta_T - (k_{\text{drop}} - \alpha) \theta_T}{\beta \theta_T + k_{\text{drop}} \sum_{t=1}^T p_t - (k_{\text{drop}} - \alpha) \sum_{t=1}^T a_t^T} = 1 + \frac{1 - \frac{1}{n}}{1 + \frac{k_{\text{drop}} \sum_{t=1}^T p_t - (k_{\text{drop}} - \alpha) \sum_{t=1}^T a_t^T}{\beta \theta_T}}$$

$$< 1 + 1 - \frac{1}{n} \text{ (Since } p_t \geq a_t^T)$$

$$= 2 - \frac{1}{n}$$

Next we prove $\sum_{t=1}^T a_t^T - \sum_{t=1}^T a_t^d \leq (n-1)\theta_T \leq n\theta_T$, $n = \lceil \frac{\beta}{(k_{\text{drop}}-\alpha)} \rceil$ by induction. For simplicity, we define $\Delta_m = \sum_{t=1}^m a_t^m - \sum_{t=1}^m a_t^d$ and $\Delta_{m,m+1} = \Delta_{m+1} - \Delta_m$. Then our inductive hypothesis is $\Delta_m \leq (n-1)\theta_m$ for $n \leq m \leq T$. Since the demand dropping threshold of $\text{ONDrop}$ is 0 for the first $n-1$ time slots, which means $a_t^d = 0$ for $1 \leq t \leq n-1$, we choose $m = n$ as the base case.

For the base case, $m = n$, the optimal threshold is $\theta_n$ according to Lemma 3, which is the $n^{th}$ largest power value in the first $n$ time slots. Clearly $a_t^n = \theta_n$ for $1 \leq t \leq n$. For $\text{ONDrop}$, $a_t^d = 0$ for $1 \leq t \leq n-1$ and $a_t^d = \theta_n$. See Figure 11 for an illustration of the base case. Therefore, $\Delta_n = \sum_{t=1}^n a_t^n - \sum_{t=1}^n a_t^d = (n-1)\theta_n$, and the inductive hypothesis holds for the base case.

Then suppose the inductive hypothesis holds for $n \leq m \leq i$, $i \leq T - 1$, which means $\Delta_m \leq (n-1)\theta_m$ holds for $n \leq m \leq i$. By Lemma 3, we know that there are at most $(n-1)$ power
values that are strictly larger than $\theta_i$ (otherwise $\theta_i$ will not be the $n^{th}$ largest power value in the first $i$ time slots).

When $m = i + 1$, the optimal threshold becomes $\theta_{i+1}$ when only the first $i + 1$ time slots are considered. If $\theta_{i+1} = \theta_i$ (only when $p_{i+1} \leq \theta_i$), both the optimal solution and ON_{Drop} will admit $p_{i+1}$ and drop 0 demand. In that case $\Delta_{i+1} = \Delta_i \leq (n-1)\theta_i = (n-1)\theta_{i+1}$, and the inductive hypothesis holds. If $\theta_{i+1} > \theta_i$ (i.e., $p_{i+1} > \theta_i$), we still have $a_{i+1}^{i+1} = a_{i+1}^d$ by Lemma 3. The non-zero area between $\theta_i$ and $\theta_{i+1}$ is $\Delta_{i,i+1}$, and $\Delta_{i,i+1} \leq (n-1)(\theta_{i+1} - \theta_i)$ since at most $(n-1)$ power values are strictly larger than $\theta_i$. Therefore $\Delta_{i+1} = \Delta_i + \Delta_{i,i+1} \leq (n-1)\theta_i + (n-1)(\theta_{i+1} - \theta_i) = (n-1)\theta_{i+1}$ and the inductive hypothesis holds.

Now we can conclude that $\Delta_m \leq (n-1)\theta_m$ for $n \leq m \leq T$, which completes our proof for the competitive ratio of ON_{Drop}. ■