HPC Data Center Participation in Demand Response: an Adaptive Policy with QoS Assurance

Yijia Zhang, *Student Member, IEEE,* Daniel C. Wilson, *Student Member, IEEE,*
Ioannis Ch. Paschalidis, *Fellow, IEEE,* and Ayse K. Coskun, *Senior Member, IEEE*

Abstract—Demand response programs help stabilize the electricity grid by providing monetary stimulus to consumers if they regulate their power consumption following market requirements. Regulation service, a market that requires participants to regulate power by following a signal updated every few seconds, is particularly beneficial to HPC data centers since data centers are capable of increasing/decreasing power consumption owing to the flexibility in running workloads and the availability of power control mechanisms. While prior works have explored how data centers can provide regulation service reserves, Quality-of-Service (QoS) provisioning for the jobs running at the data centers has not been considered. In this work, we propose an Adaptive policy with QoS Assurance that enables data centers to participate in regulation service programs with assurance on job QoS. Our policy regulates data center power through job scheduling and server power capping. QoS assurance is achieved by applying a queueing-theoretic result to our job scheduling strategy. We evaluate our policy by experiments on a real cluster. Our results demonstrate that the proposed policy reduces electricity costs by 25-56% while providing QoS assurance. On the other hand, the baseline policies cannot meet QoS constraints in 9 of the 14 workload traces tested.

Index Terms—Data center, HPC, demand response, Quality of Service, QoS assurance.

1 Introduction

Data centers[1] are playing an irreplaceable role of offering a substantial amount of computing services to society. IT companies, financial companies, and research institutes all rely on the computing power provided by innumerable data centers around the world. However, data centers are large power consumers. In 2014, all data centers in the US consumed 70 billion kWh, close to 2% of US electricity usage [1]. In 2019, the top-1 supercomputer in the world, the Summit system, consumed a peak power of 10 MW [2], equivalent to $24,000 of energy cost per day. As the trend of building large data centers is expected to continue, suppressing the increasing energy cost of data centers is a key challenge and vital to sustain their growth [1].

Participation in demand response programs is a profitable and environmentally beneficial answer to this challenge, and in this paper, we focus on regulation service, which is a specific type of demand response program [3]. In regulation service programs, a power consumer is asked to regulate its power by following a target signal broadcast by a grid operator. The power consumer determines the average power value and the range of the power according to their own regulation capability, but the exact value of the power target is not known in advance and changes every few seconds. Participants of regulation service programs benefit from significant electricity cost reduction as long as they track the target within a small error margin [3].

Data centers are good candidates to provide regulation service reserves because many data centers are capable of quickly regulating their power usage within a large range through job scheduling and server power management. Previous works have demonstrated in simulation that participation in demand response could reduce the energy cost of data centers by 50% [4], [5]. However, those works do not provide assurance on meeting the quality-of-service (QoS)[2] constraints of jobs running in data centers, which could discourage many potential participants as QoS is one of their top concerns.

To solve the problem of providing QoS guarantees in demand response, in this work, we propose an Adaptive QoS-Assurance (AQA) policy that enables data centers to participate in demand response with QoS assurance of jobs. This policy stems from the generalized processor sharing (GPS) algorithm [6], where guarantees on delay have been proven using queueing theory [7], [8]. To track the power target, our policy schedules different types of jobs according to their properties (e.g., job size, execution time, QoS constraints, etc.) and adjusts the power caps of servers. Our policy also selects the optimal bidding parameters to participate in regulation service reserves markets. The main contributions of our work are as follows:

- We propose a policy that enables data centers to participate in demand response programs with

• *Y. Zhang, D. C. Wilson, I. Ch. Paschalidis, and A. K. Coskun are with the Department of Electrical and Computer Engineering, Boston University, Boston, MA, 02215.*
E-mail: {zhangyj,danielcw,yannisp,acoskun}@bu.edu

Manuscript received April 19, 2020; revised August 26, 2020.

1. In this work, we define data centers broadly, including both enterprise and high-performance computing (HPC) data centers.

2. In this work, QoS refers to timely execution of computing jobs submitted to data centers. In other words, QoS requirement places a constraint on the delay of executing each job.

theoretically-proven guarantees on job QoS.
- We evaluate our policy by experimenting with a broad range of workload traces on a real cluster composed of enterprise-level servers.

In comparison with our recent work [9] that presents a preliminary version of our AQA policy, this paper has the following improvements: (1) Instead of considering only single-node jobs, our proposed policy now allows parallel applications that run on multiple nodes[3]. (2) The AQA policy adaptively optimizes the bidding parameters and the weight parameters used by the policy, in contrast to estimating the optimal parameters based on a single simulation. (3) Compared to our earlier experiments on a 12-server cluster, this work includes real-system experiments on a 36-server cluster with a broad set of workload settings.

Using both simulation and real-system experiments, we demonstrate that our AQA policy outperforms two baseline policies [4], [5] in terms of cost reduction and QoS assurance. We show that our policy is robust to different workload profiles, and we demonstrate that our policy reduces the electricity cost by 25-56% while providing QoS guarantees.

In the following, we first provide some background on demand response and regulation service in Section 2. Then, we describe our AQA in detail in Section 3. We elaborate our experimental methodology in Section 4 and results are presented in Section 5.

2 Background on Demand Response and Regulation Service

Due to the rapidly growing trend of incorporating renewable energy sources in the power grid [10], [11], ensuring the stability of the power grid becomes increasingly important as renewable supplies such as solar and wind are highly volatile and intermittent [12]. Various demand response programs have been developed to help stabilize the power grid by motivating the demand side of the grid to adjust power consumption in response to power supply. Peak shaving [13], dynamic energy pricing [14], and emergency load reduction [15], [16] are programs of this kind. In addition, there are demand response programs that allow the demand side to provide capacity reserves, where power consumers are required to regulate their power consumption to track a dynamic power target based on the amount of reserve that they intend to offer. By providing a larger amount of reserves, consumers receive a larger cost reduction as a reward.

There are mainly three types of capacity reserves, ordered from more to less valuable as follows: (1) *frequency control* requires power consumers to counter frequency deviations by modulating their consumption at near-real-time; (2) *regulation service* asks consumers to react to a power target broadcast by independent system operators (ISOs) every few seconds; (3) *operating reserves* are offered in a slower pace where a power target maintains its value for up to a few hours. We target regulation service in this work as it is a particularly profitable choice for data centers because data centers are capable of efficiently regulating power in a few seconds and matching the signal update interval of regulation service programs.

3. In the following, we use the two words, "node" and "server", interchangeably.

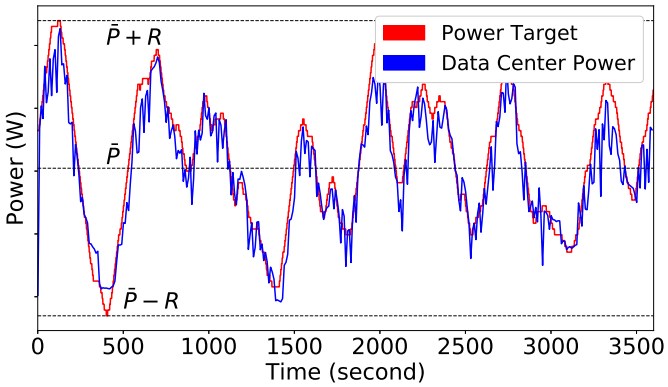


Fig. 1: The power target derived from a 1-hour ISO signal $y(t)$ and the power consumption of a data center that follows the target well. $\bar{P}$ is the average power consumption of this hour and R is the amount of reserve provided by the data center.

To participate in regulation service, at the beginning of every hour, a data center first bids for an average power consumption $\bar{P}$ and the reserve amount R according to its own power regulation capability. Then, within this hour, the power target is set as:

$$P_{target}(t) = \bar{P} + y(t)R, \tag{1}$$

where $y \in [-1, 1]$ is the signal broadcast by the ISO and is updated every few seconds. In other words, the power target $P_{target}(t)$ could change within the range of $[\bar{P} - R, \bar{P} + R]$. Although the value of signal $y(t)$ is not known in advance, it generally follows a known distribution. The signal is required to have a mean as zero, and its change rate is limited [3]. The power target derived from a 1-hour signal sample is shown in Fig. 1.

Regulation service participants should make sure their actual power follows the target closely. In this work, we express the tracking constraints in a probabilistic form [3]:

$$\text{Prob}[\epsilon(t) > 0.3] < 10\%, \tag{2}$$

where $\epsilon(t)$ is the relative tracking error defined as

$$\epsilon(t) = \frac{|P(t) - P_{target}(t)|}{R}. \tag{3}$$

In other words, power consumption must be near the target power, within a margin of 30% (with respect to R) for more than 90% of time.

At the end of this hour, the electricity bill for the data center is calculated according to its bidding parameters $\bar{P}$, R and the average tracking error $\bar{\epsilon} = \text{E}[\epsilon(t)]$. This monetary cost can be estimated by [3]

$$\text{Cost} = \left(\Pi^P \bar{P} - \Pi^R R + \Pi^\epsilon R \bar{\epsilon}\right) \times 1\text{h}, \tag{4}$$

where Π^P, Π^R, and Π^ϵ are fixed monetary cost coefficients determined by the power market.

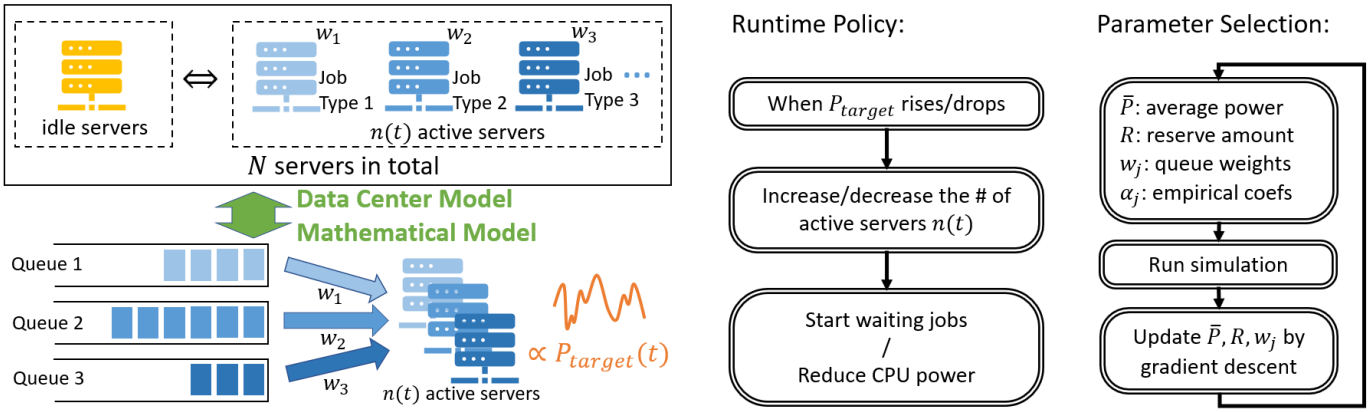


Fig. 2: Our data center model and the AQA policy. Jobs are grouped into different job types and each type is processed in a separate queue. Servers are conceptually (not physically) grouped into an idle-server group and several active-server groups. Servers in the j^{th} active group run the j^{th}-type jobs. The total number of active servers $n(t)$ is adjusted to track the power target. The Runtime Policy and the Parameter Selection algorithm constitute the AQA policy.

3 The Adaptive QoS-Assurance Policy

In this section, we first introduce our data center model and give an overview of our AQA policy. Next, we provide some background on the GPS algorithm. Then, we explain how our AQA policy regulates the total power to track the target by job scheduling and server power capping. At the end, we explain how our policy adaptively finds the optimal bidding parameters ($\bar{P}$, R) and weight parameters (w_j), when participating in the regulation service markets.

3.1 Data center and workload model

The power consumption of a data center consists of power from servers, cooling systems, and affiliated components such as network and storage systems. Because servers have a large power contribution and are typically more flexible than other systems, in this work, we focus on the regulation of server power through job scheduling and server power capping.

We assume servers are homogeneous: they consume the same amount of power when running the same job and they finish the execution of that job in the same amount of time (when under the same power cap setting). This assumption is an approximation to real data centers and is a prerequisite step towards future work considering hardware variations or data centers composed of multiple types of servers.

We group the computing jobs in a data center workload into different types according to their power consumption, processing time, and QoS constraints. We assign separate queues for different job types, and inside each queue, jobs are processed in a first-come-first-serve manner. In the following, we denote the number of job-type in a workload by J. A j^{th}-type job ($j = 1, 2, ..., J$) has processing time T^j and average power consumption p_j. We assume those values are known in advance from prior measurements. Our framework allows parallel jobs that simultaneously occupy multiple nodes to run. We assume no job consolidation, i.e., different jobs cannot share the same node, which is typical in many HPC systems due to efficiency and security considerations. By default, we assume jobs cannot be interrupted or stopped in the middle of their execution, and the relaxation of this assumption is discussed in Sections 3.6 and 5.4.

Since we have jobs grouped into different types, we also group the servers according to the jobs they are running. At runtime, the servers are dynamically and conceptually partitioned into an idle-server group and J separate active-server groups, as shown in Fig. 2. When a j^{th}-type job is started on a server, the server switches from the idle-server group into the j^{th} active-server group, and switches back after the job finishes.

We define the quality-of-service (QoS) degradation of a job as the extra time used for processing the job compared to its minimum processing time T^j_{min}, calculated by the formula:

$$Q^j = \frac{T_{so} - T^j_{min}}{T^j_{min}}. \tag{5}$$

Here, minimum processing time T^j_{min} is defined as the time for processing a j^{th}-type job without power caps and without being delayed in the queue. The sojourn time T_{so} is defined as the time from a job's submission to completion, including the waiting time in the queue T_{wait} and the actual processing time T_{proc}:

$$T_{so} = T_{wait} + T_{proc}.$$

In this paper, we focus on QoS constraints in a probabilistic form:

$$\text{Prob}[Q^j \geq Q^j_{thres}] \leq \delta^j \quad (j = 1, 2, ...), \tag{6}$$

where Q^j_{thres} is a given QoS threshold for the j^{th}-type jobs, and we set $\delta^j = 10\%$ in Sec. 4. This formula means that only a small fraction (δ^j) of j^{th}-type jobs have a QoS degradation exceeding their QoS threshold Q^j_{thres}.

3.2 An overview of our AQA policy

To provide QoS guarantees to each type of job, we partition the active servers to job types following the Generalized Processor Sharing (GPS) algorithm [6], according to which, the number of active servers allocated for each job type is proportional to a set of non-negative weights, w_j (with

TABLE 1: Variables Used in the Problem Formulation

Label	Description
R	The reserve amount to participate in regulation service
$\bar{P}$	The average power to participate in regulation service
N	The total number of servers in the data center
p_{idle}	The power consumption of a server that is idle
J	The number of job types
p_j	The power consumption of a j^{th}-type job
λ_j	The average number of j^{th}-type jobs submitted to the data center per second
m_j	The number of servers (nodes) required to run each j^{th}-type job
T^j	The processing time for each j^{th}-type job
D^j	The delay of a j^{th}-type job, i.e., the time from submission to starting
D^j_{max}	The maximal delay that the majority of j^{th}-type jobs should satisfy
Q^j_{thres}	A threshold in the QoS constraint for j^{th}-type jobs
δ^j	A probability in the QoS constraint for j^{th}-type jobs
δ^j_D	A parameter calculated by δ^j in Eq. (19)
$A_j(t)$	The random variable representing the amount of work submitted to the j^{th} queue per second
$B(t)$	The random variable representing the total amount of service provided by the data center per second
$y(t)$	The ISO signal at time t, which is always within $[-1, 1]$
$n(t)$	The number of servers that should be active at time t
α_j	An empirically-determined coefficient in quantifying the probability of large delay, in Eq. (18)
θ^*_j	A coefficient in the exponent quantifying the probability of large delay, in Eq. (18)
w_j	The weights used in the GPS algorithm

$\sum_{j=1}^{J} w_j = 1$). As we follow the GPS algorithm, a queueing-theoretic result guarantees that the delay in each queue meets QoS constraints [7], [8].

When applying our policy at runtime, at the beginning of every cycle (one cycle is one second in our experiments), the policy adjusts the total number of servers expected to be active in order to match the total power consumption with the target power. Next, the number of servers expected to be active for each job type is determined following the GPS algorithm. Then, for each job type, if the number of active servers in this group needs to increase to meet the expectation, our policy will activate idle servers to run some queued jobs of this type if there are any. On the other hand, if the number of active servers in this group needs to decrease to meet the expectation, our policy will reduce these servers' power cap instead of deactivating them because we assume no job interruption. These procedures form the runtime policy in Fig. 2.

The key to guarantee QoS and simultaneously reduce monetary cost is to select optimal bidding parameters ($\bar{P}$, R) and weight parameters (w_j). Although bidding for a larger average power $\bar{P}$ is more beneficial to guarantee QoS because a higher power target (as a result of a larger $\bar{P}$) allows more servers to run, larger $\bar{P}$ also increases the monetary cost according to Eq. (4). The weight parameters (w_j) need to be well-tuned so that a job type that is harder to meet its QoS constraint will take a larger weight and consequently, be able to access more servers. Our policy determines the optimal parameters by running simulations and applying the gradient descent on a cost function. That cost function includes both the monetary cost in Eq. (4) and an additional term penalizing QoS violation. These procedures are depicted in Fig. 2: parameter selection.

3.3 Generalized processor sharing (GPS) algorithm

GPS is an algorithm originally proposed to provide balanced performance in network scheduling [6]. It assumes that there are incoming requests[4] from several separate queues to be processed by a fixed amount of resources. The algorithm assigns a fixed non-negative weight w_j ($j = 1, 2, ...$) for each queue satisfying $\sum_j w_j = 1$, and allocates the resources to every queue according to the weights. In case a certain queue has no waiting requests, its resources will be allocated to other non-empty queues according to their weights. If we call the request-processing capability of the resources the total bandwidth, denoted as B, then the bandwidth for the j^{th} queue is at least $w_j B$, which means there exists a lower limit on the processing capability of each queue.

When extending the GPS algorithm to the scenario where the total resources $B(t)$ is not fixed but follows a stochastic process, Paschalidis et al. [7], [8] proved a theorem stating that the portion of requests that experience large delay before being processed should decrease exponentially as $m \to \infty$:

$$\text{Prob}[D^j \geq m] = \alpha_j e^{-m\theta^*_j}, \quad (j = 1, 2, ..., J). \tag{7}$$

Here, the exponential coefficients θ^*_j can be derived from the statistical properties of the requests and the processing resources.

Since the result in [7], [8] is proven only for a system with two queues, in order to accommodate a system with more queues, we follow their derivation and generalize the results for multiple-queue cases by taking a first-order assumption where different queues are assumed to be decoupled (which means that resources for an empty queue will not be shared by the non-empty queues). From our derivations in Appendix A, we prove that the portion of requests with large delay should decrease exponentially following this formula as $m \to \infty$:

$$\text{Prob}[D^j \geq m] = \alpha_j e^{-m\theta^*_j}, \quad (j = 1, 2, ..., J), \tag{8}$$

where the coefficients are calculated by

$$\theta^*_j = \sup_{\theta \geq 0,\ \Lambda_{GPS,j}(\theta) < 0} -\Lambda_B(-\theta w_j), \tag{9}$$

and where the function $\Lambda_{GPS,j}(\theta)$ is defined as

$$\Lambda_{GPS,j}(\theta) = \Lambda_{A_j}(\theta) + \Lambda_B(-\theta w_j). \tag{10}$$

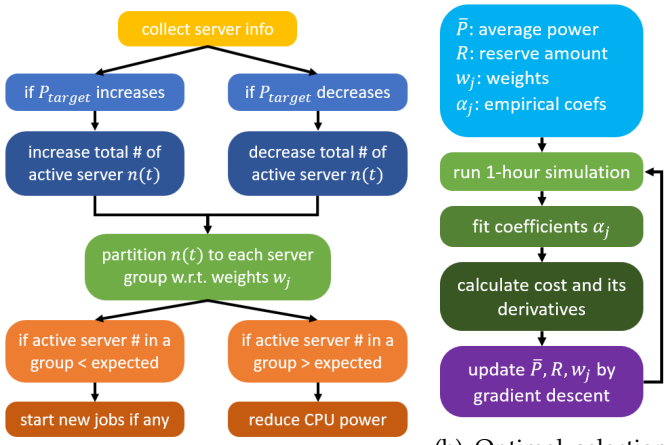
Here, Λ_{A_j} and Λ_B are the log moment-generating functions for the random variables $A_j(t)$, $B(t)$. Variable $A_j(t)$ represents the amount of requests arriving in the j^{th} queue per unit time at time t, and $B(t)$ represents the total processing capability (bandwidth) at time t; "sup" denotes the supremum of the expression under conditions.

This analytical form of large delay probability in Eq. (8) is the path towards providing theoretical guarantees on QoS in this work.

3.4 Job scheduling and power capping in AQA

Our AQA policy adjusts a data center's power consumption at runtime to match the power target P_{target} by job

4. The "requests" here are assumed to be identical in terms of their processing time.



(a) Job scheduling and server power capping at runtime.

(b) Optimal selection of bidding and weight parameters.

Fig. 3: The two components of our AQA policy.

scheduling and server power capping strategies, as shown in Fig. 3(a). The parameters used in the following are listed in Table 1.

In order to provide guarantees on job QoS, we schedule J different queues of jobs following the GPS algorithm. To match power target $P_{target}(t)$ at time t, our policy first determines the total number of active servers $n(t)$. Then, the $n(t)$ active servers are partitioned for the J queues according to their weights w_j following the GPS algorithm. As a result, we get the number of active servers for the j^{th} queue (i.e., for j^{th}-type jobs), n_j. This n_j equals to $n(t)w_j$ if all queues are non-empty, and larger if not. Next, from n_j, we get the number of j^{th}-type jobs that should be running as n_j/m_j (i.e., n_j divided by m_j), where m_j is the number of servers required for running each j^{th}-type job[5].

If n_j/m_j is larger than the current number of running jobs of the j^{th} type, then the difference between n_j/m_j and the current number of running jobs of the j^{th} type is the number of j^{th}-type jobs that should be scheduled to start at this moment. On the other hand, if n_j/m_j is smaller than the current number of running jobs of the j^{th} type, because we do not want to terminate jobs before they finish, we reduce data center power by reducing the server power caps of all active servers, which is discussed in the following paragraphs. Whenever reducing the server power caps is not necessary, our policy always let servers run without power caps.

From the policy described above, we see that the total number of active servers $n(t)$ at time t should be determined by matching the target power with the data center's power consumption, i.e.:

$$P_{target} = \bar{P} + y(t)R = (N - n(t))p_{idle} + \sum_{j=1}^{J} n_j p_j,$$

which is equivalent to

$$n(t) = \frac{\bar{P} + y(t)R - p_{idle}N}{\left(\sum_{j=1}^{J} w_j p_j\right) - p_{idle}}. \tag{11}$$

5. We call a job as a single-server job if $m_j = 1$, and a multi-server job or parallel job if $m_j > 1$.

Here, N is the total number of servers in the data center. p_{idle} is the idle server power, and p_j (also denoted as $p_{j,max}$ later) is the power for running a j^{th}-type job without power capping. We have also made an approximation $n_j = n(t)w_j$.

Our AQA policy applies server power capping only in situations when the already-running jobs are consuming more power than the target. When that happens, our policy reduces the power cap on all the active servers by the same ratio $\gamma \in [0, 1]$ to ensure fairness. To be more specific, for a j^{th}-type job whose power consumption is $p_{j,max}$ and $p_{j,min}$ when under the highest/lowest server power cap, we apply a server power cap that makes the job running at power $p_{j,cap}$ defined by the equation

$$\gamma = \frac{p_{j,cap} - p_{j,min}}{p_{j,max} - p_{j,min}}.$$

The ratio γ is determined by matching the target power and the actual consumption:

$$P_{target} = (N - n(t))p_{idle} + \sum_{j=1}^{J} n_j p_{j,cap}.$$

3.5 Bidding and weight parameter selection in AQA

Our policy determines the optimal selection of bidding parameters $\bar{P}$, R, and weights w_j by solving the following optimization problem:

$$\min_{\bar{P},R,w_j} \left(\Pi^P \bar{P} - \Pi^R R + \Pi^\epsilon R\bar{\epsilon}\right) \times 1\text{h} \tag{12}$$

$$\text{subject to} \quad \text{Prob}[Q^j \geq Q^j_{thres}] \leq \delta^j, \quad j = 1, 2, ... \tag{13}$$

$$\sum_{j=1}^{J} w_j = 1, \quad \bar{P}, R, w_j > 0. \tag{14}$$

In the following, we first simplify the QoS constraints in Eq. (13) using the queueing-theoretical result in Sec. 3.3. To start with, we combine Eq. (13) and Eq. (5), and we get

$$\text{Prob}[Q^j \geq Q^j_{thres}] \leq \delta^j \tag{15}$$

$$\Leftrightarrow \quad \text{Prob}\left[\frac{T^j_{wait} + T^j_{proc} - T^j_{min}}{T^j_{min}} \geq Q^j_{thres}\right] \leq \delta^j \tag{16}$$

$$\Leftrightarrow \quad \text{Prob}[T^j_{wait} \geq Q^j_{thres} T^j_{min}] \leq \delta^j. \tag{17}$$

Here, Eq. (16) is transformed into Eq. (17) by approximation since the actual processing time T^j_{proc} is usually close to the minimum T^j_{min}. Combining Eq. (8) with Eq. (17) leads to

$$\Leftrightarrow \quad \text{Prob}[D^j \geq D^j_{max}] = \alpha_j e^{-D^j_{max}\theta^*_j} \leq \delta^j \tag{18}$$

$$\Leftrightarrow \quad \theta^*_j \geq \delta^j_D = -\frac{1}{D^j_{max}} \ln\left(\frac{\delta^j}{\alpha_j}\right). \tag{19}$$

Converting Eq. (17) into Eq. (18) is merely a change of notation in order to match the notation in Eq. (8).

To further simplify Eq. (19) using Eq. (9), we quantify the statistical properties of job arrival and power target. Because different types of jobs have different power consumption and processing times, a job cannot be simply regarded as a "request" in Sec. 3.3. Instead, we convert jobs and servers into the unit of "amount of service". A j^{th}-type job, using m_j servers to run and with a minimum processing time of T^j_{min}, is considered as requiring $m_j T^j_{min}$ amount of

service. As a result, if we assume the job arrival for this queue follows a Poisson process with parameter λ_j (i.e., the average number of j^{th}-type jobs arriving per unit time), then, the amount of service injected to the j^{th} queue per unit time, $A_j(t)$, follows a Poisson process. The log moment-generating function for $A_j(t)$ is

$$\Lambda_{A_j}(\theta) = \lambda_j(e^{\theta m_j T_j} - 1). \tag{20}$$

Similarly, $n(t)$ active servers at time t are considered as having a processing capability of $B(t) = n(t)$ amount of service per unit time. Since the matching of power target with data center power gives us the relation Eq. (11), the statistical property of $n(t)$ depends on the property of the signal $y(t)$. Regulation service programs require the average value over a long time $\bar{y}$ to be close to 0. From the ISO signal sample we have, we empirically determine that the signal $y(t)$ generally follows a normal distribution, whose standard deviation is estimated as $y_\sigma = 0.40$. As a consequence, $n(t)$ follows a normal distribution with an average value

$$n_\mu = \frac{\bar{P} - p_{idle}N}{\left(\sum_{j=1}^{J} w_j p_j\right) - p_{idle}}, \tag{21}$$

and a standard deviation

$$n_\sigma = \frac{y_\sigma R}{\left(\sum_{j=1}^{J} w_j p_j\right) - p_{idle}}. \tag{22}$$

Thus, the log moment-generating function of $B(t)$ is

$$\Lambda_B(\theta) = n_\mu\theta + \frac{1}{2}n_\sigma^2\theta^2. \tag{23}$$

Eqs. (9)(10)(20)(23) provide us the θ_j^* defined in Eq. (19), and as we have seen, satisfying Eq. (19) provides the QoS assurance we need.

Although θ_j^* can be derived from the statistic properties of job arrival and ISO signal, the coefficient α_j can only be estimated empirically. In our previous work [9], we obtained a fixed estimate of this coefficient by running one experiment and fitted it with the observed QoS-degradation probability using Eq. (18). However, because the fixed estimate of α_j using a specific set of parameters ($\bar{P}$, R, and w_j) could have errors at other $\bar{P}$, R, w_j values, in this work, we improve this procedure by adaptively adjusting the estimation of α_j while optimizing the cost function using gradient descent.

In order to apply the gradient descent optimization, we make the QoS-assurance constraint (Eq. (13)) as a part of the cost function (Eq. (12)), and we estimate the cost of tracking error as C_{error}. Then, the cost function becomes

$$\begin{aligned} C &= \left(\Pi^P\bar{P} - \Pi^R R\right)H + C_{error} \\ &\quad +\beta\sum_j \text{SoftPlus}\left(\rho\left(\text{Prob}[Q^j - Q^j_{thres}] - \delta^j\right)\right) \end{aligned} \tag{24}$$

Here, H represents 1 hour. Function SoftPlus(x) is defined as $\ln(1 + e^x)$, which is a smooth approximation of the ramp function $\max(0, x)$. Therefore, the QoS-related term in Eq. (24) is close to zero when the QoS constraint is met, and positive when violated. Parameters β and ρ control whether the QoS constraints are less or more strict. Although larger β and ρ result in more strict constraints, they also make the surface of cost function steeper, and consequently, finding the optimum becomes harder.

To calculate the derivatives of the tracking error cost, we need an analytical estimation of the tracking error cost:

$$C_{error} = \Pi^\epsilon R\bar{\epsilon}H \tag{25}$$

$$= \Pi^\epsilon \int_0^H |P_{target}(t) - P_{actual}(t)|dt \tag{26}$$

$$\simeq \Pi^\epsilon \int_0^H (\bar{P} + y(t)R - P_{actual}(t))dt \tag{27}$$

$$= \Pi^\epsilon \left[\bar{P}H - E_{actual}\right] \tag{28}$$

$$\simeq \Pi^\epsilon H \left[\bar{P} - p_{idle}N - \sum_j \lambda_j m_j T_j \left(p_j - p_{idle}\right)\right] \tag{29}$$

Here, H again represents 1 hour. Eq. (25) is the definition of tracking error cost, and Eq. (26) is from the definition of the average tracking error. Eq. (27) removes the absolute sign because this tracking error term is significant only when $\bar{P} + y(t)R \gg P_{actual}(t)$, and the other case where $\bar{P} + y(t)R \ll P_{actual}(t)$ is precluded by the QoS-related term in Eq. (24) because a small $\bar{P}$ already violates QoS significantly. In Eq. (28), E_{actual} represents the actual energy consumption in that 1 hour, which is estimated in Eq. (29) following our assumption of job arrivals according to Poisson processes.

Applying Eqs. (18)(20-24)(29), we can compute the derivatives of the cost function, $\frac{\partial C}{\partial \bar{P}}$, $\frac{\partial C}{\partial R}$, $\frac{\partial C}{\partial w_j}$, to be used in gradient descent optimization. More details on calculating the derivatives are in Appendix B.

Figure 3(b) shows the algorithm we apply to perform the optimization. The algorithm starts with a set of initial values for parameters $\bar{P}$, R, w_j, and α_j, and runs a one-hour[6] simulation. Next, parameter α_j is estimated for each jobtype j by fitting the QoS degradation curve with Eq. (18). Then, we calculate the cost function with its derivatives, and update $\bar{P}$, R, w_j through gradient descent[7]. These new parameters are fed into simulation again and we iterate over the process above. In this optimization approach, the time complexity of performing the theoretical calculations is constant irrespective of the data center size, and in our experiments, the time spent on theoretical calculations is negligible (less than 1 second). The time for conducting simulation depends on the data center size. In our experiments, this optimization approach always finds a solution meeting all constraints in less than 200 iterations, which takes no more than a few minutes even for a large data center with 10k nodes.

3.6 Additional methods to regulate power

In the sections above, we have explained how our policy regulates power consumption to match the target by job scheduling and server power capping. The flexibility of our policy allows additional methods to address potential non-ideal situations. Two typical non-ideal situations include:

6. Simulating a one-hour running time of the 36-node data center takes less than 5 seconds using our simulator.

7. Because there is a constraint $\sum_j w_j = 1$ in our optimization problem, we apply gradient descent with projection.

Long execution time: When jobs with long execution times are running but the target power decreases sharply, the data center power may not be able to match the target because long-running jobs do not finish quickly and server power capping alone may not be able to reduce a large portion of power. To handle this situation, our policy can further apply *job preemption*, which interrupts the long-running jobs temporarily, and only resume their execution when target power matching is achievable. When applying job preemption, our policy prioritizes selecting jobs that are least possible to violate their QoS constraints, and the number of jobs preempted are determined by matching the power target. When the power target increases, resuming preempted jobs is prioritized before starting new jobs, and jobs closer to QoS violation are prioritized to resume.

Lack of jobs: When there are not enough jobs waiting in the queues but the target power is high, the data center power may not be able to match the target as servers are mostly in idle state due to this lack of jobs. To handle this situation, our policy can work with a queue of *standby jobs* and start jobs in this standby-job-queue. The standby-job-queue can be composed of jobs that have no QoS constraint or have a relatively loose QoS constraint at the time-scale of days. When other queues are empty, our policy starts jobs from this standby-job-queue, and the number of jobs to start is determined by matching the estimated total power consumption with the power target.

One real-life example for the standby-job-queue is the *overrun queue* in the Cori system at the National Energy Research Scientific Computing Center (NERSC) [17]. The overrun queue allows Cori users running out of their CPU-hour quota to submit jobs to this queue without monetary costs. Even though the overrun queue is convenient for the users, there is no guarantee on the waiting time in the overrun queue, and a job there may also be terminated by the system after running for 4 hours. Another similar example is the Amazon EC2 Spot Instances which offer spare computing capacity in the Amazon cloud at a steep discount [18]. Users can enjoy the low price of this service at the cost of unexpected interruptions: the cloud service provider may interrupt the service with a warning two minutes in advance whenever there are no sufficient spare servers.

4 Experimental Methodology

We implement our AQA policy and evaluate it using both simulation and real-system experiments. Fig. 4 shows the architecture of our implementation. Among all the servers, we choose one server to be the "master" that receives the ISO signal, applies our AQA policy, and controls job scheduling and power capping. The other servers are called "clients" and they communicate with the master frequently (once per second in our experiments) to receive a control message and send their job/power status. We implement the communication between the master and clients using *rabbitmq* [19].

A controller process in each client server executes the power capping of the server. It reads the power consumption from sensors and accordingly sets a cap on the CPU power to match the control message sent from the master. In our system, that controller process is a PID controller with parameters $P = 0.4$, $I = 0$, $D = 0$. We determine these parameters by running benchmark applications under a changing powercap and selecting the parameters that shows the quickest response and the best stability.

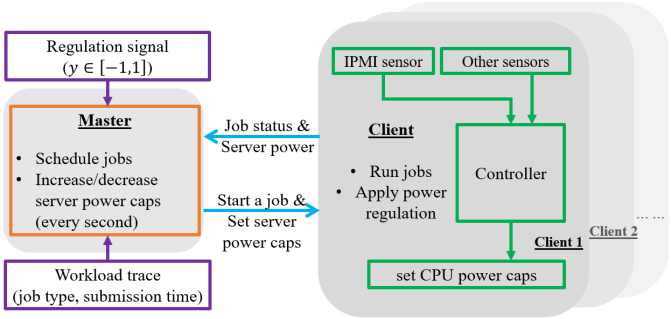


Fig. 4: Our real-system implementation architecture.

4.1 System setup

We use 36 servers from the Boston University Shared Computing Cluster (BU-SCC) that are physically located at the Massachusetts Green High Performance Computing Center (MGHPCC). Each server has two Intel Xeon Gold 6132 processors. Each processor has 14 cores, and its thermal design power is 140 W. When conducting experiments on this cluster, we use one server as the master, and the other 35 as clients.

We use the Linux *perf* tool [20] to read the power of CPU and memory in a server. We also use the IPMI tool [21] to read an entire server's power, which includes the power from CPU, memory, disk, fan, network interface, motherboard, etc. Because the IPMI reading is a running-average value of the server power and it has a large granularity of 4 W, we fit a power model based on the *perf* reading to obtain the real-time server power and increase the granularity. We fit the power model by running benchmark applications and collecting the CPU/memory power (P_{proc} and P_{mem}) from *perf* and server power (P_{server}) from IPMI. We find that a linear model, $P_{server} = \phi_1 P_{proc} + \phi_2 P_{mem} + \phi_3$, is accurate enough for our purpose, and we empirically determine $\phi_1 = 1.29$, $\phi_2 = 1.63$, $\phi_3 = 44.0$ W in our experiments.

4.2 Workload profile

We generate 14 different workload traces using parallel applications from the NAS Parallel Benchmark (NPB) suite [22]. These benchmarks allow a few different inputs and can be processed using different number of threads/servers. Because modifying the input and the number of threads/servers significantly changes the processing time and power consumption, in our evaluation, we use the word "application" or "a type of job" to refer to a benchmark with a specific input and processed with a specific number of threads on a specific number of servers. For example, application `bt.C.16` means running benchmark `bt` with input C and with 16 threads, and we run it on 1 server, as shown in Table 2.

Table 2 shows the properties of the applications and the composition of the 14 workload traces: W1∼W14. The applications in each trace are marked in the table. Column

TABLE 2: Applications and workloads used in evaluation. The meaning of application name is shown by this example: `bt.C.16` means running benchmark `bt` with input C and with 16 threads. Here, m_j is the size (number of servers used to run). T_{min} (T_{max}) is the minimum (maximum) processing time in seconds and p_{max} (p_{min}) is the corresponding power consumption of a server in Watts.

App	m_j	T_{min}	p_{max}	T_{max}	p_{min}	Q_{thres}	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
bt.C.16	1	73	339	86	249	2.5		✓					✓	✓		✓				✓
bt.C.25	1	53	402	70	276	4.7		✓					✓							
mg.D.16	1	84	380	105	266	2.8				✓			✓		✓	✓				
sp.C.16	1	54	375	62	267	7.1		✓		✓			✓		✓		✓			
ep.D.88	4	40	360	53	237	6.3			✓											
is.D.32	3	42	249	42	241	5.6	✓			✓			✓	✓			✓	✓	✓	
bt.C.36	2	38	343	46	249	3.1						✓	✓	✓		✓			✓	
bt.D.49	2	551	391	729	250	5.6	✓				✓		✓		✓		✓	✓	✓	
ep.D.64	3	54	353	70	237	3.9				✓			✓		✓					
sp.D.100	4	343	399	381	264	3.3			✓		✓		✓		✓					
lu.D.224	8	89	400	119	250	7.6			✓								✓			
ep.D.28	1	124	383	175	238	5.9					✓				✓					
cg.C.4	1	28	238	28	239	4.0				✓				✓						
bt.D.25	1	1022	402	1370	254	3.2		✓			✓		✓		✓	✓				
lu.D.28	1	763	429	954	270	5.0						✓								
mg.D.8	1	141	297	151	258	2.9	✓	✓			✓		✓	✓		✓		✓	✓	
sp.D.16	1	1165	355	1302	270	5.5		✓					✓				✓			
is.D.4	1	122	204	123	194	7.3	✓				✓							✓	✓	
cg.D.16	1	743	326	823	253	7.3		✓					✓	✓			✓			
ep.D.56	2	64	383	90	238	2.0														✓
ft.D.64	3	284	321	313	247	3.1							✓			✓				
ft.D.128	6	165	321	179	242	7.7			✓				✓							
sp.D.196	8	329	370	352	253	3.7			✓											
lu.C.28	1	29	413	43	255	6.9		✓		✓										
cg.D.128	6	231	336	242	246	4.0			✓		✓			✓						
cg.D.32	3	364	281	390	246	5.5					✓			✓			✓			
ep.D.100	4	36	366	49	238	4.5	✓		✓							✓		✓	✓	
is.D.64	4	27	287	28	228	3.1	✓		✓	✓								✓		
lu.D.112	4	164	405	222	251	4.1	✓									✓		✓	✓	
mg.D.32	2	49	378	58	266	5.0	✓			✓			✓		✓		✓	✓	✓	
sp.C.64	3	31	371	32	258	2.2														✓

m_j is jobsize that represents the number of servers we use to run a certain application. When running an application, T_{min} (T_{max}) is the minimum (maximum) processing time in seconds and p_{max} (p_{min}) is the corresponding power consumption of a server in Watts. We determine the values of these parameters by running experiments on our 36-server cluster. Column Q_{thres} is the QoS threshold defined in Eq. (6), whose values are randomly generated within 2 to 7.9.

For each workload trace, the applications in the trace are selected with some randomness while following a trace-specific rule summarized in Table 3. In addition, 8 applications are selected in W1∼W5 and W8∼W13. Cases with less/more types of applications are explored by W6 and W7. We also assume a 50% data center utilization level for W1∼W11 and W14. Cases with lower/higher utilization level are explored by W12 and W13.

We generate a workload trace by generating the job arrival time of each application to follow a Poisson process with arrival rate λ_j. These arrival rates are related to the data center utilization level η by an approximate equation:

$$\sum_{j=1}^{J} \lambda_j T_{j,min} = \eta N.$$

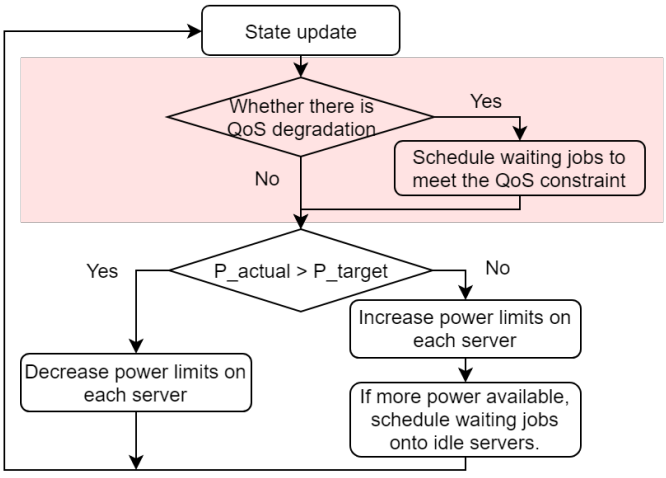


Fig. 5: Simplified flow chart illustrating the two baseline policies. EnergyQARE [5] or Tracking-only policy [4] corresponds to the chart with or without the pink region, respectively.

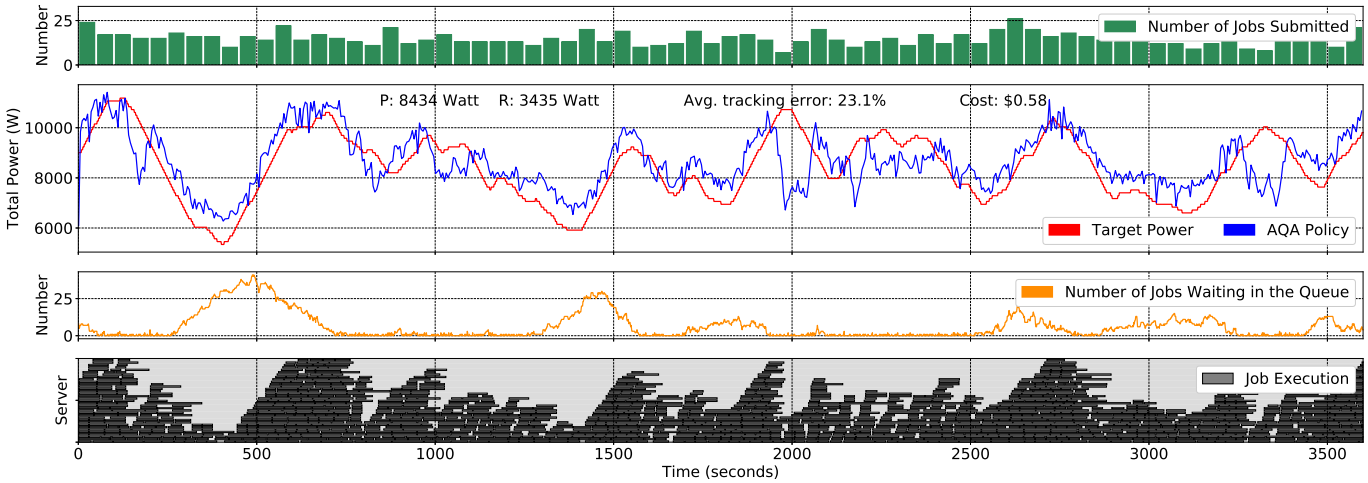


Fig. 6: Experiments on a real 36-server cluster running workload W4 using our AQA policy.

TABLE 3: Workload trace properties.

Trace	Property	Note
W1	Mix	Applications cover a wide range of size, processing time, power, and QoS threshold
W2	Single	Only contains single-server applications
W3	LargeSize	Applications of size 4∼8 servers
W4	ShortTime	Applications with $T_{min} \leq 120$ s
W5	LongTime	Applications with 122 s $\leq T_{min} \leq 1022$ s
W6	LessType	Contains 2 types of applications
W7	MoreType	Contains 16 types of applications
W8	LowPower	Applications with $p_{max} \leq 350$ W
W9	HighPower	Applications with $p_{max} > 350$ W
W10	TightQoS	Applications with $Q_{thres} \leq 5$
W11	SlackQoS	Applications with $Q_{thres} > 5$
W12	Util-25%	Average system utilization is 25%
W13	Util-90%	Average system utilization is 90%
W14	ExtraTight	Top 3 applications with the smallest Q_{thres}

Assuming each application shares the data center utilization equally, we can derive the arrival rate for type-j jobs as

$$\lambda_j = \frac{\eta N}{T_{j,min} J}. \tag{30}$$

It deserves mentioning that our assumption of Poisson-distribution job arrival and normal-distribution ISO signal is only a special application of our policy for the experimental evaluation in this work, and our AQA policy can also be applied for other distributions of job arrivals and ISO signal.

For simulation, we build a simulator following the same architecture as the one shown in Fig. 4. Our simulator models simulated servers whose behaviors are close to the BU-SCC servers in our real-system experiments. When a simulated server is idle, we assume it consumes 169 W power, which is the average idle server power in our real system. When the simulated servers are running an application, we assume the power they consume and the time they take to finish the application follow the power and time values in Table 2. To be specific, when an application is run with p_{max} (or p_{min}) power, it takes T_{min} (or T_{max}) time to finish. If the power cap of running an application is set to be a value p_{cap} within the range of (p_{min}, p_{max}), then we assume the application finishes in T time in simulation. Here, T satisfies

$$\frac{T_{max} - T}{T_{max} - T_{min}} = \frac{p_{cap} - p_{min}}{p_{max} - p_{min}}.$$

This formula follows a first-order approximation to the actual power-performance relation.

4.3 Baseline policies

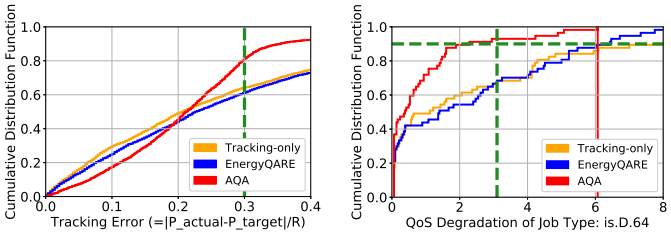
We compare our AQA policy with two baselines proposed in previous works: the Tracking-only policy [4] and the EnergyQARE policy [5]. The flow chart in Fig. 5 with or without the pink region corresponds to the EnergyQARE or the Tracking-only policy. Obviously, the Tracking-only policy focuses entirely on tracking the target and it ignores job QoS. EnergyQARE has an additional QoS-aware block (shown in pink) which is activated whenever the relative QoS degradation, defined as Q^j/Q^j_{thres}, is larger than the relative tracking error, defined as $\epsilon/0.3$. This QoS-aware block tries to schedule waiting jobs as more as possible and the power target constraint is ignored temporarily. To select parameters $\bar{P}$ and R in EnergyQARE, we do a grid search in the valid range of $\bar{P}, R$ by running simulations, and we choose the best $\bar{P}, R$ that minimize QoS degradation and tracking error. We use the same $\bar{P}, R$ from EnergyQARE to run the Tracking-only policy.

5 Results

To conduct real-system experiments, we run each workload trace and each policy (AQA, Tracking-only, and EnergyQARE) for one hour on our cluster. We are not applying job preemption and standby-job-queue in Sections 5.1-5.2. Later in Section 5.3, we present the results of AQA policy with standby jobs. In Section 5.4, we present the results of AQA policy with job preemption. Section 5.5 presents comparison between different QoS levels, and Section 5.6 presents results for a 10k-node data center. All results presented in Sections 5.1-5.3 are from experiments on our real system. Results in Sections 5.4-5.6 are from simulation.

5.1 A 36-server real-system experiment

Figure 6 shows a typical result for running the AQA policy on our 36 servers. The workload trace for this experiment is W4. The green bars show the number of jobs (summed over all types) submitted to the queues in each time interval. The red curve represents the target power calculated by Eq. (1)



(a) Tracking error. (b) QoS degradation.

Fig. 7: The cumulative distribution functions (CDF) of the tracking error and QoS degradation of the three policies.

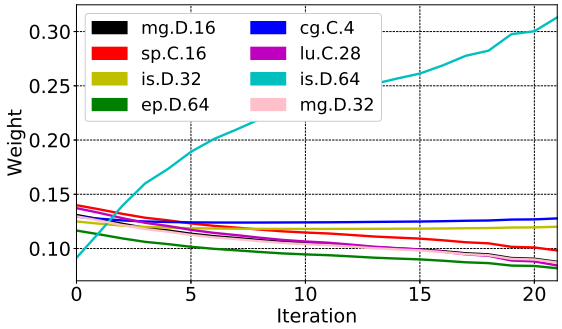


Fig. 8: Weights adjusted by gradient-descent optimization.

from the ISO signal. The blue curve represents the total power consumption of the 35 client servers. We see clearly that the real power consumption follows closely with the target power.

The yellow curve shows the total number of jobs waiting in the queues. Clearly, this waiting job number is negatively correlated with the target power. At $t = 400$ s and 1400 s, when the target power drops, the number of active servers needs to be diminished, and consequently, more jobs are held in the waiting state. The deep grey stripes at the bottom of Fig. 6 show the period when a server is running a job, and the vertical displacement of a stripe denotes the server's index number.

In this experiment, the estimated electricity cost for these servers calculated by Eq. (4) is \$0.58 when applying our AQA policy. On the other hand, the electricity cost for applying the Tracking-only or EnergyQARE policy is \$0.77 or \$0.78, 33% higher than AQA (see Appendix C Fig. 1 for the results of the EnergyQARE policy). Figure 7 compares the cumulative distribution functions (CDFs) of the tracking-error violation and the QoS violation[8] among the three policies. This proves that our AQA policy provides more cost reduction than the baselines without violating tracking and QoS constraints. If not participating in demand response, the electricity cost can be estimated as $\Pi^P \bar{P} \times 1\text{h} =$ \$0.84. By comparing the cost when participating in demand response, \$0.58, with the cost when not participating in demand response, \$0.84, we conclude that AQA policy can reduce electricity cost by 31% while abiding by QoS constraints.

8. Due to space limits, only one type of job's QoS degradation is drawn here, and the other 7 types are shown in Appendix C Fig. 2.

TABLE 4: Experiments with 14 workload traces using the AQA policy.

Trace	Cost	Cost Reduction
W1	\$0.58	30%
W2	\$0.57	37%
W3	\$0.68	29%
W4	\$0.58	31%
W5	\$0.34	56%
W6	\$0.64	34%
W7	\$0.58	33%
W8	\$0.56	29%
W9	\$0.58	37%
W10	\$0.58	34%
W11	\$0.46	46%
W12	\$0.46	38%
W13	\$0.71	31%
W14	\$0.64	33%

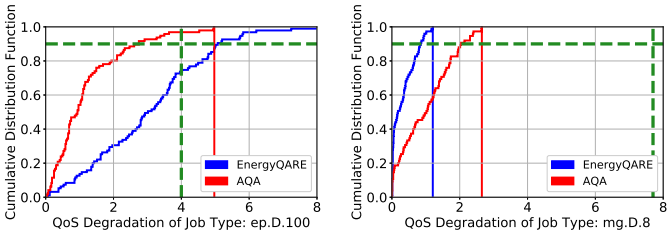
The optimal $\bar{P}$ and R for AQA policy with this workload are 8434 W and 3435 W, which are selected following the gradient-descent-based method discussed in Sec. 3.5. Figure 8 shows how the weights for the 8 types of application are adjusted through iterations. Although all weights are similar at initialization, the weight for application `is.D.64` increases significantly to 31.3% at the end, meanwhile the weights for other applications decrease slightly to save space for `is.D.64` as all weights should sum to 1. The application `is.D.64` needs a much larger weight than others because of its relatively large size (taking 4 servers), short processing time (27 s), and relatively strict QoS constraint (Q_{thres}=3.1).

This capability of fine-tuning the weights for different applications by gradient-descent optimization is one of the key advantages of our AQA policy over the EnergyQARE policy. In EnergyQARE, all applications are treated similarly in using servers, so the jobs that are easier to violate their QoS constraint cannot gain higher priority. Our AQA policy, instead, balances the QoS of all types of applications by optimizing the cost function in Eq. (24). Through iterations of simulation and gradient-descent optimization, any application type that suffers from a high QoS violation probability will be given a higher weight by reducing the weights of other applications. If the weights of other applications are already at critical values and cannot be further reduced, $\bar{P}$ will increase and R will decrease to provide more space in tuning the weights.

5.2 Results from 14 different workload traces

We experiment with the 14 workload traces listed in Table 3, and our findings are summarized as follows:

- With workload trace W2, all three policies meet both the QoS constraints and the tracking constraint.
- With workload traces W6, W8, W10, and W11, AQA and EnergyQARE can meet all constraints, whereas the Tracking-only policy violates QoS constraints.
- With workload trace W1, W3∼W5, W7, W9, W12∼W14, only the AQA policy meets all constraints (standby jobs needed for W3, see Section 5.3), whereas the Tracking-only policy and the EnergyQARE policy violate either the QoS constraints or the tracking constraint.



(a) Application `ep.D.100`. (b) Application `mg.D.8`.

Fig. 9: The cumulative distribution functions of QoS degradation when running W13 with AQA or EnergyQARE.

In summary, our experiments with the 14 workloads prove that our proposed AQA policy outperforms EnergyQARE and Tracking-only as AQA meets QoS and tracking constraints in all tested workloads while EnergyQARE and Tracking-only cannot. The results also prove that Tracking-only is the worst among the three policies at meeting QoS objectives as Tracking-only can only meet the QoS constraints for 1 out of 14 workloads. Although Tracking-only policy performs well at tracking the regulation signal, it sacrifices QoS too much.

Table 4 summarizes the cost and cost reduction of AQA policy in these experiments. Here, the cost with demand response participation is calculated following Eq. (4), and the cost without demand response participation is estimated as $\Pi^P \bar{P} \times 1\text{h}$. The cost reduction is calculated by comparing the costs with and without demand response participation. Our experiment results show that AQA policy reduces the electricity cost by 29-56%.

The single-server workload mix, W2, performs well in all evaluated policies. This matches the results of the evaluations in the prior works that proposed the EnergyQARE and the Tracking-only policies [4], [5]. Intuitively, single-server jobs are much better than multi-server jobs in regulation service participation because single-server jobs are easier for scheduling while larger jobs need to wait until the time when enough servers are available. In addition, even though all three policies perform well in trace W2, AQA is better than the other two as AQA achieves the lowest electricity cost: \$0.57, which is 7% (or 8%) lower than the \$0.61 (or \$0.62) from Tracking-only (or EnergyQARE).

It also deserves mentioning the result from workload trace W13 as it is a good example demonstrating the benefits of the adaptive optimization of weights in AQA policy. For this workload trace, AQA can meet both tracking and QoS constraints. EnergyQARE either fails to meet the tracking error constraint or fails to meet the QoS constraints of some applications even after we exhaustively experiment with all valid pairs of $\bar{P}, R$ in simulation. Figure 9(a) compares the cumulative distribution functions of the QoS degradation of application `ep.D.100` in AQA or EnergyQARE[9], and Fig. 9(b) is for application `mg.D.8`. These figures show that the AQA policy enables both `ep.D.100` and `mg.D.8` to meet their QoS constraints. On the other hand, using EnergyQARE, `ep.D.100` fails to meet its QoS constraint while `mg.D.8`'s QoS is too good and even better than its QoS in AQA policy.

A main reason behind this is that `mg.D.8` runs on 1 server but `ep.D.100` takes 4 server, so `mg.D.8` is easier to be scheduled due to its small size, resulting in its low QoS degradation in EnergyQARE. However, the failure of `ep.D.100` to meet its QoS constraints in EnergyQARE proves that we need a mechanism to give different priorities to these two applications so that we can sacrifice the performance of `mg.D.8` to improve the performance of `ep.D.100`. Therefore, the reason why AQA enables both applications to meet their constraints is the adaptive optimization of weights. In fact, the gradient-descent-based optimization in AQA results in a 38.8% weight for `ep.D.100` and a 4.2% weight for `mg.D.8`.

To verify that our policy can be applied to QoS constraint values other than the specific Q_{thres} values shown in Table 2, we also conduct simulations for 10 times with randomized selection of Q_{thres} values within the range from 2 to 7.9. In all cases, we find that our optimization method is able to find weights and bidding parameters that guarantee all jobs to meet their QoS constraints while participating in demand response.

5.3 AQA with standby jobs

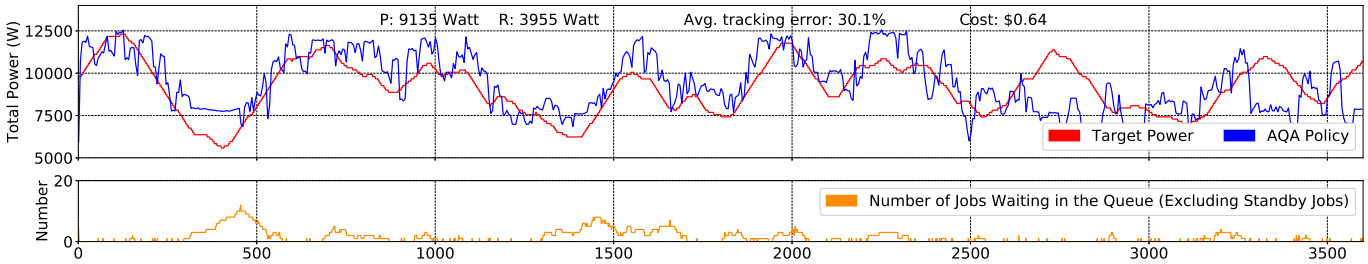
As discussed in Section 3.6, we suggest a standby job queue to solve the potential problem caused by lack of submitted jobs. In case that regular jobs are large in size, standby jobs also help improve the power tracking performance if their sizes are smaller.

In our experiments with workload trace W3 where each regular job takes 4 to 8 servers (which is considered "large" as they already occupy 11% to 22% of our cluster), applying our AQA policy without standby jobs results in violation of tracking error (see Fig. 10(a)) because large jobs are much less flexible in scheduling. The average tracking error here is 30.1%. On the other hand, with standby jobs of one server in size, Fig. 10(b) shows better tracking performance where the average tracking error is 14.7%. The standby jobs will start whenever there are insufficient jobs. As a result, AQA with standby jobs smooths the power consumption curve and enables the power consumption to match the target at $t = 2700$, instead of leaving a 3000 W gap at $t = 2700$ in Fig. 10(a) due to the lack of waiting jobs.

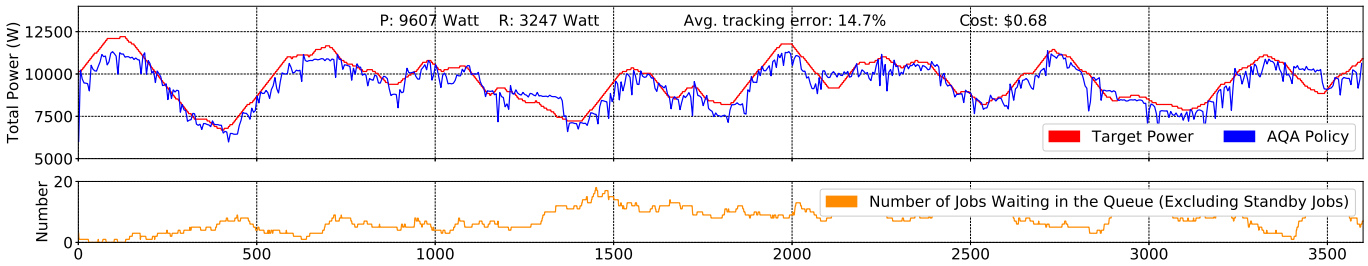
5.4 AQA with job preemption

We evaluate job preemption in simulation using workload trace W5, where applications have relatively long execution time ranging from 122 s to 1022 s. Figure 11 compares the results of the AQA policy with or without job preemption. The same $\bar{P}$, R, and weights are used. The green regions in Fig. 11(b) represent the time where a job is preempted to reduce power and resumed later. As a result, the total power consumption at $t = 400$, $t = 1400$, $t = 1800$, etc. in job preemption case is lower than the case without job preemption, enabling AQA with job preemption to achieve a lower average tracking error of 6.2%, in comparison with the 9.5% tracking error from AQA without job preemption. Both cases in Fig. 11 meet the tracking error constraint (according to their CDFs curves, not shown here), so the real benefit

9. In this figure, EnergyQARE runs with a best pair of $\bar{P}, R$ that has the lowest QoS degradation while meeting tracking error constraint.

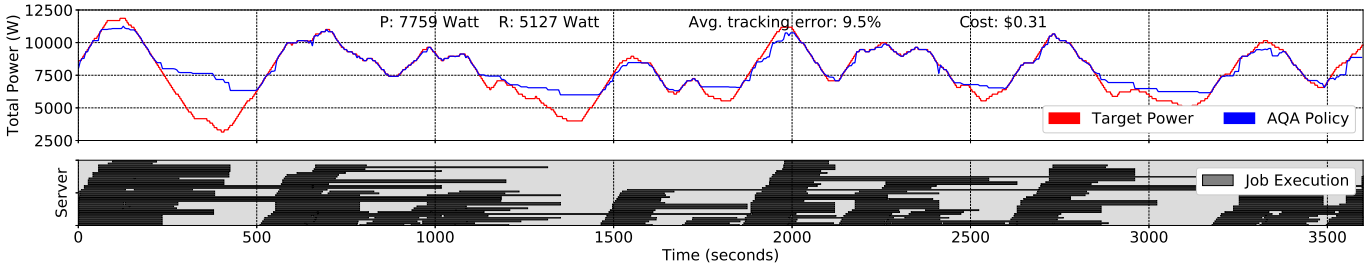


(a) AQA policy without standby jobs.

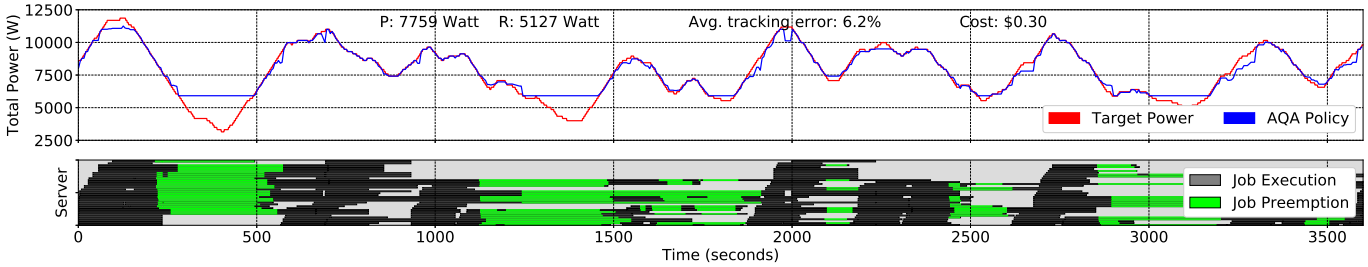


(b) AQA policy with standby jobs.

Fig. 10: Experiments on a real 36-server cluster running workload trace W3 using our AQA policy.



(a) AQA policy without job preemption.



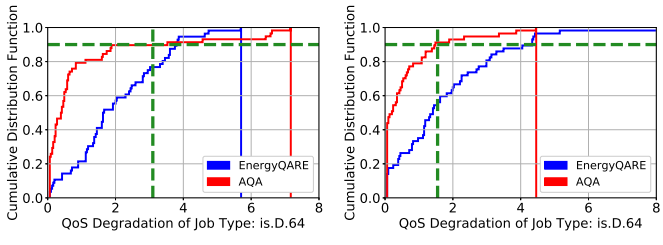
(b) AQA policy with job preemption.

Fig. 11: Evaluating AQA policy with/without job preemption by simulation using workload trace W5.

of job preemption here is a slight decrease of electricity cost from $0.31 to $0.30. However, if using some workload traces that have application execution time even longer than the ones in W5, we find that AQA without job preemption may not be able to meet the tracking error constraint. In those cases, job preemption becomes necessary.

5.5 Comparison of different QoS constraint levels

To evaluate the impact of different QoS constraint levels on our policy, we conduct simulations with a workload when applications' QoS constraints are changed from tight to loose. Starting from the QoS constraints shown in Table 2 column Q_{thres} (we call them as in a "medium" QoS



(a) Medium QoS level. (b) Tight QoS level.

Fig. 12: Comparing results from workload W4 with either a medium or a tight QoS constraint level.

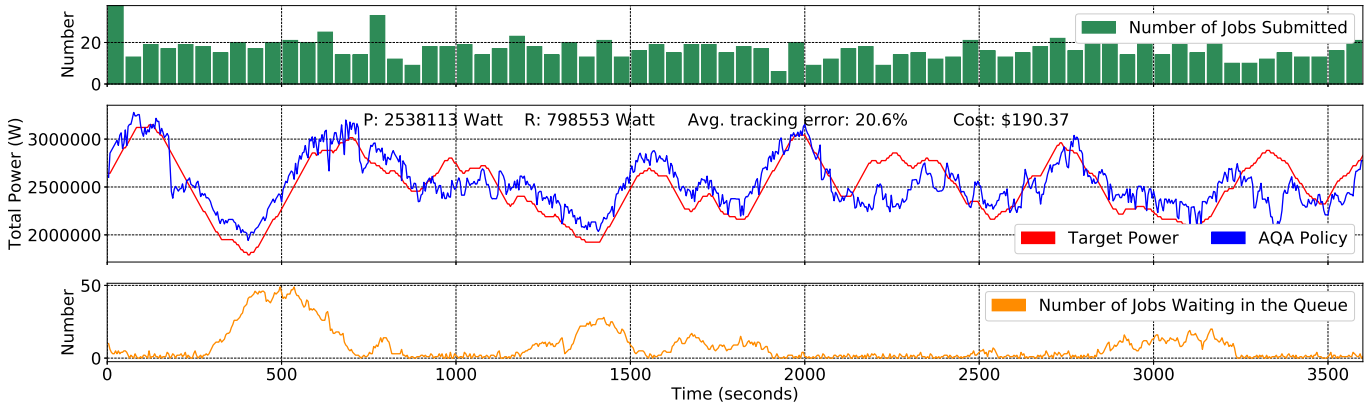


Fig. 13: Evaluating AQA policy by simulating a 10k-node data center with a job-size-scaled version of W1 workload.

constraint level), we tighten the QoS constraints of the applications by decreasing the Q_{thres} of each application by half, and we loosen the QoS constraints of the applications by doubling the Q_{thres} of each application. These simulations do not apply a standby job queue or job preemption. Figure 12 compares the simulation results of workload W4 with either a medium or a tight QoS constraint level. As we can see, when the QoS constraint levels change from medium in Fig. 12(a) to tight in Fig. 12(b), both our AQA policy and the EnergyQARE policy improve the QoS of this application, and our policy enables the application to meet the QoS constraint while EnergyQARE policy does not. We only show the QoS degradation curve of application `is.D.64` due to the space limit, but the results for other applications have similar behaviors as this one. Therefore, we conclude that our AQA policy has the opportunity to meet either a loose or a tight QoS constraint assuming there is a valid solution that can be achieved by selecting the appropriate values for the weights and bidding parameters.

5.6 Scalability to large data centers

To evaluate the scalability of our policy when applied to large data centers, we conduct simulation studies with a data center composed of ten thousand servers. Figure 13 shows the simulation results of running a job-size-scaled version of workload W1 for one hour. Here, the workload is composed of the same types of applications in Table 2 (W1) but the number of nodes used by each application is scaled up by 100x (e.g., application `is.D.32` takes 300 nodes instead of 3 nodes), and the job arrival rates are also adjusted following Eq. (30) to match an assumed 50% utilization level of this large data center. As shown in Fig. 13, our policy works well for a large data center with 10k-node and enables the actual power to follow the target power closely. We also check the QoS degradation of the applications, and they all meet their QoS constraints. The electricity cost with demand response participation in this experiment is $190.4 according to Eq. (4). Without demand response participation, the electricity cost for running these jobs can be estimated as $\Pi^P \bar{P} \times 1\text{h} = \$253.8$. Therefore, the cost reduction for this experiment is 25%. If the data center has similar power consumption and cost reduction throughout the year, then it can save $555,822 ($= 253.8 \times 25\% \times 24 \times 365$) per year by participation in demand response with our policy.

6 Related Work

The problem of improving efficiency for data centers has received considerable attentions. According to a recent survey [23], several major HPC centers around the world are employing or are exploring energy-aware resource management strategies. It has been reported that these HPC centers are integrating job schedulers with power grid information, developing power-adaptive scheduling in SLURM, detecting power-hungry applications at runtime, or building power-capping infrastructures [23].

In recent years, there have been significant advances on integrating data centers and HPC systems with power markets. To enable data centers to participate in power markets, various power programs including peak shaving [13], dynamic energy pricing [14], and emergency load reduction [15], [16] have been explored. Different kinds of policies have been proposed for data centers to participate in frequency control, regulation service, or operating reserves [4], [5], [24], [25], [26], [27]. The impact of power limitation on the performance of data centers is also important. Using a model inspired by a real HPC system, Borghesi et al. [28] showed that it is possible to apply frequency scaling to save energy without penalizing users. Another recent trend is exploring the coordination of multiple data centers together in power markets. Some strategies are proposed to enable multiple data centers to collaborate in power markets and mitigate workload uncertainty [29], [30].

In conjunction with these efforts, there has been a growing interest for data centers to participate in demand response. Prior works have explored strategies for data centers to participate in demand response by joint management of IT workloads with cooling facilities [25], [31], [32], renewable energy sources [33], [34], energy storage devices [34], [35], or electric vehicles [36]. It has also been shown that without renewable energy sources or energy storage devices, computing servers are capable of adjusting power consumption to meet the power target in demand response. Chen et al. [4] developed a heuristic power regulation policy for data centers to participate in regulation service though job scheduling, processor power capping, and server state transition. Chen et al. [5] also designed a QoS-aware policy for data centers to meet regulation service requirement while considering jobs' QoS. The evaluation of data center integration with power market is usually done

in simulation. Recently, a few works (including our previous work [9]) evaluated policies on small-scale real systems [9], [37].

In addition to works from the perspective of data centers, there are also works from the perspective of power markets. Clausen et al. presented a qualitative study of service contracts between electricity service providers and data centers in the United States and Europe [38]. Novel incentive mechanisms have been proposed to motivate power grids and data centers to participate in demand response [39], [40], [41]. For geo-distributed colocation data centers, Sun et al. proposed an online auction mechanism for emergency demand response to motivate data centers to shuffle workload across multiple sites [42].

7 Conclusion

Data centers are large power consumers, but the flexibility of job scheduling and the capability of server power capping enable them to regulate their total power consumption at a short time-scale. The data centers' capability to significantly regulate their power renders them particularly good candidates for demand response programs, especially the regulation service where the participant needs to follow a target power that changes every few seconds. Prior works developed policies that enable data centers to participate in regulation service but without providing assurances on the QoS of computing jobs [4], [5]. To fill this gap, in this work, we propose an Adaptive policy with QoS Assurance (AQA) that provides service QoS guarantees based on queueing theoretic results. We implemented and evaluated our policy on a real cluster composed of 36 enterprise-level servers by running a broad set of workload traces. We demonstrated that our AQA policy meets both tracking error constraints and job QoS constraints, outperforming two earlier policies which cannot meet the constraints in some cases.

Potential future directions can focus on relaxing the assumptions we use. First, because our policy assumes the knowledge of the distributions of job arrival processes, it will be interesting to explore what happens if the distributions deviate from the assumed models. If there are periodic trends or other predictable features in job arrival patterns, a predictor could be added to improve the performance in that case. Second, since our policy assumes the knowledge of jobs' expected execution time and power usage, it will be useful if this need for accurate knowledge can be replaced by some strategies that only need approximate knowledge.

Acknowledgments

The authors would like to thank Michael Dugan from Boston University Research Computing Services group and Naved Ansari from Massachusetts Open Cloud for their help in reserving servers for our experiments. Research was partially supported by the NSF under grants IIS-1914792, DMS-1664644, and CNS-1645681, by the NIH under grant R01 GM135930, by the ONR under grant N00014-19-1-2571, and by the Boston University College of Engineering under the Dean's Catalyst Award.

References

[1] A. Shehabi, S. Smith, N. Horner, I. Azevedo, R. Brown, J. Koomey, E. Masanet, D. Sartor, M. Herrlin, and W. Lintner, "United states data center energy usage report," *Lawrence Berkeley National Laboratory, Berkeley, California. LBNL-1005775 Page*, vol. 4, 2016.

[2] TOP500. (2019) The top 500 list. [Online]. Available: https://www.top500.org/lists/2019/11/

[3] New York Independent System Operator (NYISO), "Ancillary services manual, v6.0," NYISO, Manual, May 2020. [Online]. Available: https://www.nyiso.com/manuals-tech-bulletins-user-guides

[4] H. Chen, M. C. Caramanis, and A. K. Coskun, "The data center as a grid load stabilizer," *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, no. i, pp. 105–112, 2014.

[5] H. Chen, Y. Zhang, M. C. Caramanis, and A. K. Coskun, "Energyqare: Qos-aware data center participation in smart grid regulation service reserve provision," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, no. 1, pp. 2:1–2:31, Jan. 2019.

[6] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, Jun. 1993.

[7] I. C. Paschalidis, "Class-specific quality of service guarantees in multimedia communication networks," *Automatica*, vol. 35, no. 12, pp. 1951 – 1968, 1999.

[8] D. Bertsimas, I. C. Paschalidis, and J. N. Tsitsiklis, "Large deviations analysis of the generalized processor sharing policy," *Queueing Systems*, vol. 32, no. 4, pp. 319–349, Nov 1999.

[9] Y. Zhang, I. C. Paschalidis, and A. K. Coskun, "Data center participation in demand response programs with quality-of-service guarantees," in *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, ser. e-Energy '19, 2019, p. 285–302.

[10] C. Bohringer, A. Loschel, U. Moslener, and T. F. Rutherford, "Eu climate policy up to 2020: An economic impact assessment," *Energy Economics*, vol. 31, Supplement 2, pp. S295 – S305, 2009.

[11] EIA, "Annual energy outlook 2014," http://www.eia.gov/forecasts/aeo, 2014.

[12] C. Novoa and T. Jin, "Reliability centered planning for distributed generation considering wind power volatility," *Electric Power Systems Research*, vol. 81, no. 8, pp. 1654 – 1661, 2011.

[13] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, "Benefits and limitations of tapping into stored energy for datacenters," in *Proceedings of the 38th International Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2011, pp. 341–352.

[14] T. N. Le, Z. Liu, Y. Chen, and C. Bash, "Joint capacity planning and operational management for sustainable data centers and demand response," in *Proceedings of the 7th International Conference on Future Energy Systems*, ser. e-Energy '16, 2016, pp. 16:1–16:12.

[15] L. Zhang, S. Ren, C. Wu, and Z. Li, "A truthful incentive mechanism for emergency demand response in colocation data centers," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 2632–2640.

[16] N. H. Tran, C. Pham, S. Ren, Z. Han, and C. S. Hong, "Coordinated power reduction in multi-tenant colocation datacenter: An emergency demand response study," in *ICC*, May 2016, pp. 1–6.

[17] National Energy Research Scientific Computing Center (NERSC). (2020) NERSC Queue Policy. [Online]. Available: https://docs.nersc.gov/jobs/policy/

[18] Amazon. (2020) Amazon ec2 spot instances. [Online]. Available: https://aws.amazon.com/ec2/spot

[19] B. Chapman *et al.*, "Rabbitmq," https://github.com/rabbitmq, 2018.

[20] D. Melo and A. Carvalho, "The new linux perf tools," in *Slides from Linux Kongress*, vol. 18, 2010.

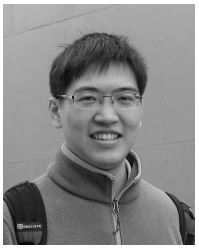
[21] D. Laurie *et al.*, "An open-source tool for controlling ipmi-enabled systems," https://github.com/ipmitool/ipmitool, 2018.

[22] D. H. Bailey, E. Barszcz, J. T. Barton *et al.*, "The nas parallel benchmarks," *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, pp. 158–165, 1991.

[23] M. Maiterth, G. Koenig, K. Pedretti, S. Jana, N. Bates, A. Borghesi, D. Montoya, A. Bartolini, and M. Puzovic, "Energy and power aware job scheduling and resource management: Global survey — initial analysis," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2018, pp. 685–693.

[24] W. Wang, A. Abdolrashidi, N. Yu, and D. Wong, "Frequency regulation service provision in data center with computational flexibility," *Applied Energy*, vol. 251, p. 113304, 2019.

[25] T. Cioara, I. Anghel, M. Bertoncini, I. Salomie, D. Arnone, M. Mammina, T. Velivassaki, and M. Antal, "Optimized flexibility management enacting data centres participation in smart demand response programs," *Future Generation Computer Systems*, vol. 78, pp. 330 – 342, 2018.
[26] B. Aksanli and T. Rosing, "Providing regulation services and managing data center peak power budgets," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–4.
[27] K. Ahmed, J. Liu, and X. Wu, "An energy efficient demand-response model for high performance computing systems," in *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sep. 2017, pp. 175–186.
[28] A. Borghesi, A. Bartolini, M. Milano, and L. Benini, "Pricing schemes for energy-efficient hpc systems: Design and exploration," *The International Journal of High Performance Computing Applications*, vol. 33, no. 4, pp. 716–734, 2019.
[29] Z. Yu, Y. Guo, and M. Pan, "Coalitional datacenter energy cost optimization in electricity markets," in *Proceedings of the Eighth International Conference on Future Energy Systems*, ser. e-Energy '17. New York, NY, USA: ACM, 2017, pp. 191–202.
[30] L. Niu and Y. Guo, "Enabling reliable data center demand response via aggregation," in *Proceedings of the Seventh International Conference on Future Energy Systems*, ser. e-Energy '16. New York, NY, USA: ACM, 2016, pp. 22:1–22:11.
[31] L. Cupelli, T. Schütz, P. Jahangiri, M. Fuchs, A. Monti, and D. Müller, "Data center control strategy for participation in demand response programs," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 5087–5099, Nov 2018.
[32] Z. Chen, L. Wu, and Z. Li, "Electric demand response management for distributed large-scale internet data centers," *IEEE Transactions on Smart Grid*, vol. 5, no. 2, pp. 651–661, 2014.
[33] T. N. Le, Z. Liu, Y. Chen, and C. Bash, "Joint capacity planning and operational management for sustainable data centers and demand response," in *Proceedings of the Seventh International Conference on Future Energy Systems*, ser. e-Energy '16. New York, NY, USA: ACM, 2016, pp. 16:1–16:12.
[34] A. Pahlevan, M. Zapater, A. Coskun, and D. Atienza, "Ecogreen: Electricity cost optimization for green datacenters in emerging power markets," *IEEE Transactions on Sustainable Computing*, pp. 1–1, 2020.
[35] Y. Shi, B. Xu, B. Zhang, and D. Wang, "Leveraging energy storage to optimize data center electricity cost in emerging power markets," in *Proceedings of the Seventh International Conference on Future Energy Systems*, ser. e-Energy '16. New York, NY, USA: ACM, 2016, pp. 18:1–18:13.
[36] S. Li, M. Brocanelli, W. Zhang, and X. Wang, "Integrated power management of data centers and electric vehicles for energy and regulation market participation," *IEEE Transactions on Smart Grid*, vol. 5, no. 5, pp. 2283–2294, 2014.
[37] J. McClurg, R. Mudumbai, and J. Hall, "Fast demand response with datacenter loads," in *2016 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, Sep. 2016, pp. 1–5.
[38] A. Clausen, G. Koenig, S. Klingert, G. Ghatikar, P. M. Schwartz, and N. Bates, "An analysis of contracts and relationships between supercomputing centers and electricity service providers," in *Proceedings of the 48th International Conference on Parallel Processing: Workshops*, ser. ICPP 2019, 2019, pp. 4:1–4:8.
[39] I. C. Paschalidis, B. Li, and M. C. Caramanis, "Demand-side management for regulation service provisioning through internal pricing," *IEEE Transactions on Power Systems*, vol. 27, no. 3, pp. 1531–1539, 2012.
[40] Z. Zhou, F. Liu, S. Chen, and Z. Li, "A truthful and efficient incentive mechanism for demand response in green datacenters," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2018.
[41] Y. Wang, F. Zhang, C. Chi, S. Ren, F. Liu, R. Wang, and Z. Liu, "A market-oriented incentive mechanism for emergency demand response in colocation data centers," *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 13 – 25, 2019.
[42] Q. Sun, S. Ren, C. Wu, and Z. Li, "An online incentive mechanism for emergency demand response in geo-distributed colocation data centers," in *Proceedings of the Seventh International Conference on Future Energy Systems*, ser. e-Energy '16. New York, NY, USA: ACM, 2016, pp. 3:1–3:13.



Yijia Zhang received the BS degree from the Department of Physics, Peking University, Beijing, China. He is working toward the PhD degree in the Department of Electrical and Computer Engineering, Boston University. His research interests include high performance computing, computer system optimization, and machine learning.

Daniel Curtis Wilson Daniel Curtis Wilson received the BS degrees in Computer Science and Computer Engineering from NC State University, Raleigh, North Carolina. He is working toward the PhD degree in Computer Engineering at Boston University. Prior to his current studies, Daniel worked at NetApp and Itron. He worked as an intern at Intel while pursuing his PhD. His current research interests include energy-aware computing and systemwide optimization.

Ioannis Ch. Paschalidis (M'96–SM'06–F'14) received the Diploma in ECE from the National Technical University of Athens, Athens, Greece, in 1991, and the M.S. and Ph.D. degrees, both in EECS, from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 1993 and 1996, respectively.

In September 1996 he joined Boston University where he has been ever since. He is a Professor and Data Science Fellow at Boston University with appointments in the Department of Electrical and Computer Engineering, the Division of Systems Engineering, the Department of Biomedical Engineering, and the Faculty of Computing & Data Sciences. He is the Director of the Center for Information and Systems Engineering (CISE). He has held visiting appointments with MIT and Columbia University, New York, NY, USA. His current research interests lie in the fields of systems and control, networks, applied probability, optimization, operations research, computational biology, and medical informatics.

Dr. Paschalidis is a recipient of the NSF CAREER award (2000), several best paper and best algorithmic performance awards, and a 2014 IBM/IEEE Smarter Planet Challenge Award. He was an invited participant at the 2002 Frontiers of Engineering Symposium, organized by the U.S. National Academy of Engineering and the 2014 U.S. National Academies Keck Futures Initiative (NAFKI) Conference. From 2013 to 2019 he was the founding Editor-in-Chief of the IEEE Transactions on Control of Network Systems.

Ayse K. Coskun received the MS and PhD degrees in Computer Science and Engineering from the University of California, San Diego. She is a Professor with the Department of Electrical and Computer Engineering, Boston University (BU). She was with Sun Microsystems (now Oracle), San Diego, prior to her current position at BU. Her research interests include energy-efficient computing, computer architecture, embedded systems, and management and optimization of large-scale computing systems. She serves as an associate editor of IEEE Transactions on CAD and IEEE Transactions on Computers.

APPENDIX A
DERIVATION OF EQ. (9)

We derive Eq. (9) using Theorem 7 in Ref. [7] and Theorem 7.2 in Ref. [8].

Theorem 7 in Ref. [7] proves that, in a two-class system under the GPS policy, as $m \to \infty$, the delay tail probability can be approximated by

$$\text{Prob}[D^j \geq m] \approx \alpha_j e^{-m\theta_j^*}, \quad (j = 1, 2).$$

Eq. (28) in Ref. [7] provides

$$\theta_1^* = \sup_{\theta \geq 0,\ \Lambda_{GPS,1}(\theta) < 0} \left[\Lambda_{A^1}(\theta) - \Lambda_{GPS,1}(\theta)\right],$$

where $\Lambda_{GPS,1}$ is defined by

$$\Lambda_{GPS,1} = \max\left[\Lambda^{\text{I}}_{GPS,1}(\theta), \Lambda^{\text{II}}_{GPS,1}(\theta)\right],$$

and $\Lambda^{\text{I}}_{GPS,1}(\theta)$, $\Lambda^{\text{II}}_{GPS,1}(\theta)$ are defined in Eqs. (22)(23) in Ref. [7]. Because $\Lambda^{\text{I}}_{GPS,1}(\theta)$ corresponds to the case where the second queue is empty and the effective bandwidth of the first queue is larger than $w_1 B(t)$, we neglect this case following our decoupling assumption. This implies $\Lambda_{GPS,1} = \Lambda^{\text{II}}_{GPS,1}(\theta)$, and

$$\theta_1^* = \sup_{\theta \geq 0,\ \Lambda_{GPS,1}(\theta) < 0} \left[\Lambda_{A^1}(\theta) - \Lambda^{\text{II}}_{GPS,1}(\theta)\right]. \tag{31}$$

In the proof of Theorem 7.2 in Ref. [8], $\Lambda^{\text{II}}_{GPS,1}(\theta)$ is given by

$$\Lambda^{\text{II}}_{GPS,1}(\theta) = \sup_a \sup_{\substack{x_1 - w_1 x_3 = a \\ x_2 \geq w_2 x_3}} [\theta a - \Lambda^*_{A^1}(x_1) - \Lambda^*_{A^2}(x_2) - \Lambda^*_B(x_3)],$$

where $\Lambda^*(\cdot)$ is the Legendre transform of $\Lambda(\cdot)$, defined by

$$\Lambda^*(a) = \sup_\theta \left(\theta a - \Lambda(\theta)\right).$$

$\Lambda(\cdot)$ denotes the log-moment generating functions, and $x_1(t)$, $x_2(t)$, $x_3(t)$ are the empirical rates of the random process A^1, A^2, B, respectively. Because of the decoupling assumption, the influence of the process A^2 can be removed from Eq. (32). Consequently, we obtain

$$\begin{aligned}
\Lambda^{\text{II}}_{GPS,1}(\theta) &= \sup_a \sup_{x_1 - w_1 x_3 = a} [\theta a - \Lambda^*_{A^1}(x_1) - \Lambda^*_B(x_3)] \\
&= \sup_{x_1} \sup_{x_3} [\theta x_1 - \theta w_1 x_3 - \Lambda^*_{A^1}(x_1) - \Lambda^*_B(x_3)] \\
&= \sup_{x_1} [\theta x_1 - \Lambda^*_{A^1}(x_1) + \Lambda_B(-\theta w_1)] \\
&= \Lambda_{A^1}(\theta) + \Lambda_B(-\theta w_1). \qquad (32)
\end{aligned}$$

Combining Eq. (31) and Eq. (32), we conclude at

$$\theta_1^* = \sup_{\theta \geq 0,\ \Lambda_{GPS,1}(\theta) < 0} -\Lambda_B(-\theta w_1). \tag{33}$$

For a multi-queue system, because of our decoupling assumption, Eq. (33) holds for the first queue. Because all queues are equivalent to each other, we can directly generalize Eq. (33) by symmetry, and arriving at Eq. (9).

APPENDIX B
CALCULATING THE DERIVATIVES IN EQ. (24)

We show the derivation of $\frac{\partial C}{\partial \bar{P}}$. The other derivatives including $\frac{\partial C}{\partial R}$ and $\frac{\partial C}{\partial w_j}$ can be derived similarly.

Starting with Eq. (24), we have

$$\begin{aligned}
\frac{\partial C}{\partial \bar{P}} &= \Pi^P H + \Pi^\epsilon H + \\
&\quad \beta \sum_j \left(\frac{e^{\rho(\text{Prob}[Q^j - Q^j_{thres}] - \delta^j)}}{1 + e^{\rho(\text{Prob}[Q^j - Q^j_{thres}] - \delta^j)}} \times \right. \\
&\quad \left. \rho \frac{\partial}{\partial \bar{P}} \text{Prob}[Q^j - Q^j_{thres}] \right). \qquad (34)
\end{aligned}$$

Using Eq. (18), we obtain

$$\frac{\partial}{\partial \bar{P}} \text{Prob}[Q^j - Q^j_{thres}] = \alpha_j e^{-D^j_{max}\theta_j^*}(-D^j_{max}) \frac{\partial \theta_j^*}{\partial \bar{P}}. \tag{35}$$

To calculate the derivatives of θ_j^*, we first calculate θ_j^* by plugging Eq. (23) into Eq. (9), which yields

$$\begin{aligned}
&\sup_{\theta \geq 0,\ \Lambda_{GPS,j}(\theta) < 0} -\Lambda_B(-\theta w_j) \qquad (36) \\
= &\sup_{\theta \geq 0,\ \Lambda_{GPS,j}(\theta) < 0} -n_\mu(-\theta w_j) - \frac{1}{2} n_\sigma^2 (-\theta w_j)^2 \\
= &\sup_{\theta \geq 0,\ \Lambda_{GPS,j}(\theta) < 0} -\frac{1}{2} n_\sigma^2 w_j^2 \left(\theta - \frac{n_\mu}{n_\sigma^2 w_j}\right)^2 + \frac{n_\mu^2}{2n_\sigma^2} \qquad (37)
\end{aligned}$$

Here, whether the maximum point $\frac{n_\mu^2}{2n_\sigma^2}$ of the above quadratic function can be reached depends on whether $\theta = \frac{n_\mu}{n_\sigma^2 w_j}$ meets the conditions $\theta \geq 0,\ \Lambda_{GPS,j}(\theta) < 0$.

To evaluate these conditions, we plug Eqs. (20)(23) into Eq. (10) and obtain

$$\Lambda_{GPS,j}(\theta) = \lambda_j (e^{\theta m_j T_j} - 1) - n_\mu \theta w_j + \frac{1}{2} n_\sigma^2 \theta^2 w_j^2.$$

As the second-order derivative $\Lambda''_{GPS,j}(\theta)$ is always positive, the function $\Lambda_{GPS,j}(\theta)$ is convex. Since 0 is a root of $\Lambda_{GPS,j}(\theta)$, the other root will be positive if and only if

$$\begin{aligned}
&\Lambda'_{GPS,j}(\theta)\big|_{\theta=0} < 0 \qquad (38) \\
\Leftrightarrow \quad &w_j > \frac{\lambda_j m_j T_j}{n_\mu} \qquad (39) \\
\Leftrightarrow \quad &n_\mu w_j > \lambda_j m_j T_j. \qquad (40)
\end{aligned}$$

Equation (40) is actually the requirement that the average computing service provided to the j-th queue should be larger than the average amount of work submitted to this queue. Thus, we can safely assume Eq. (40) is satisfied, otherwise the queue length will diverge and the QoS constraint will be violated. Therefore, there exists a positive root for $\Lambda_{GPS,j}(\theta)$, and whether $\theta = \frac{n_\mu}{n_\sigma^2 w_j}$ meets the conditions $\Lambda_{GPS,j}(\theta) < 0$ can be converted to

$$\begin{aligned}
&\Lambda_{GPS,j}\left(\frac{n_\mu}{n_\sigma^2 w_j}\right) < 0 \\
\Leftrightarrow \quad &\lambda_j (e^{\frac{n_\mu m_j T_j}{n_\sigma^2 w_j}} - 1) < \frac{n_\mu^2}{2n_\sigma^2} \\
\Leftrightarrow \quad &w_j > \frac{n_\mu m_j T_j}{n_\sigma^2 \ln\left(1 + \frac{n_\mu^2}{2n_\sigma^2 \lambda_j}\right)}. \qquad (41)
\end{aligned}$$

In the following, we separately discuss the two cases depending on whether Eq. (41) is satisfied or not.

Case I: If Eq. (41) holds, from Eq. (37) we get

$$\theta_j^* = \sup_{\theta \geq 0,\ \Lambda_{GPS,j}(\theta)<0} -\Lambda_B(-\theta w_j) = \frac{n_\mu^2}{2n_\sigma^2}. \tag{42}$$

From Eq. (21), we get

$$\frac{\partial n_\mu}{\partial \bar{P}} = \frac{1}{\left(\sum_{j=1}^{J} w_j p_j\right) - p_{idle}}. \tag{43}$$

To summarize, in Case I, combining Eqs. (34)(35)(42)(43) provides us the derivative $\frac{\partial C}{\partial \bar{P}}$ we need.

Case II: If Eq. (41) does not hold, i.e.,

$$w_j \leq \frac{n_\mu m_j T_j}{n_\sigma^2 \ln\left(1 + \frac{n_\mu^2}{2n_\sigma^2 \lambda_j}\right)} \tag{44}$$

assuming Θ_j is the positive root of $\Lambda_{GPS,j}(\theta)$, i.e.

$$\Lambda_{GPS,j}(\Theta_j) = \Lambda_{A_j}(\Theta_j) + \Lambda_B(-\Theta_j w_j) = 0 \tag{45}$$

then the supremum in Eq. (36) is achieved at Θ_j, i.e.,

$$\begin{aligned} \theta_j^* &= \sup_{\theta \geq 0,\ \Lambda_{GPS,j}(\theta)<0} -\Lambda_B(-\theta w_j) \\ &= -\Lambda_B(-\Theta_j w_j) \\ &= \Lambda_{A_j}(\Theta_j) \\ &= \lambda_j(e^{\Theta_j m_j T_j} - 1). \end{aligned}$$

Then, we have

$$\frac{\partial \theta_j^*}{\partial \bar{P}} = \lambda_j e^{\Theta_j m_j T_j} m_j T_j \frac{\partial \Theta_j}{\partial \bar{P}}. \tag{46}$$

To calculate $\frac{\partial \Theta_j}{\partial \bar{P}}$, we take derivative $\frac{\partial}{\partial \bar{P}}$ on both sides of Eq. (45), and after re-arrangement, we get

$$\frac{\partial \Theta_j}{\partial \bar{P}} = \frac{\frac{\partial n_\mu}{\partial \bar{P}} \Theta_j w_j - n_\sigma \frac{\partial n_\sigma}{\partial \bar{P}} \Theta_j^2 w_j^2}{\lambda_j e^{\Theta_j m_j T_j} m_j T_j - n_\mu w_j + n_\sigma^2 \Theta_j w_j^2}. \tag{47}$$

From Eq. (22), we have

$$\frac{\partial n_\sigma}{\partial \bar{P}} = 0. \tag{48}$$

To summarize, in Case II, combining Eqs. (34)(35)(43)(45-48) provides us the derivative $\frac{\partial C}{\partial \bar{P}}$ we need.

APPENDIX C
ADDITIONAL FIGURES

Figure 14 show the results of applying EnergyQARE policy with workload W4 on our cluster. Figure 15 compares the cumulative distribution function of three policies, Tracking-only, EnergyQARE, and AQA.

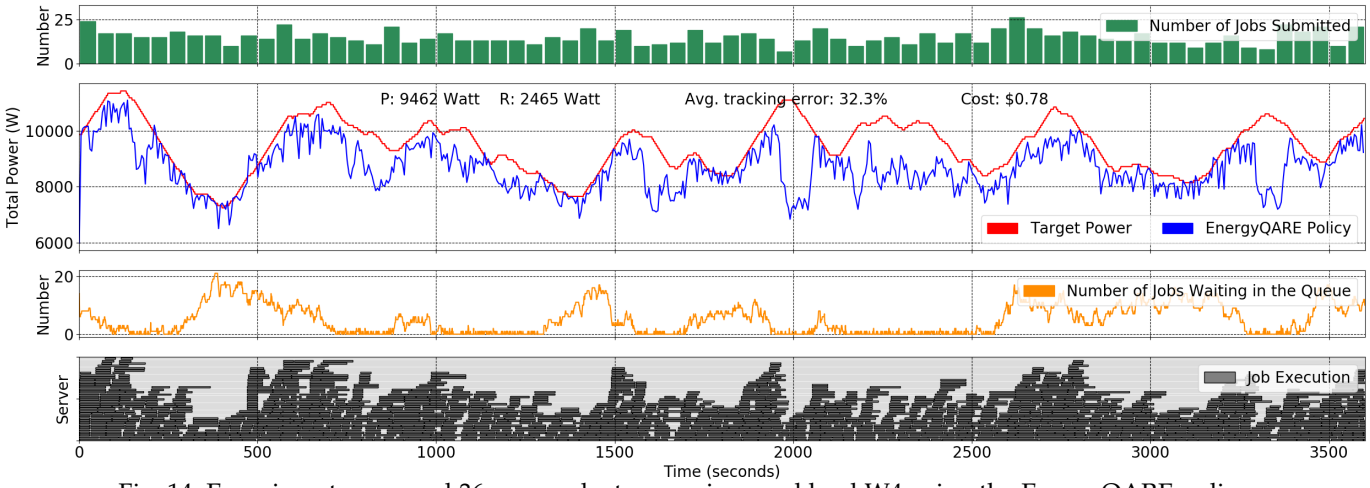
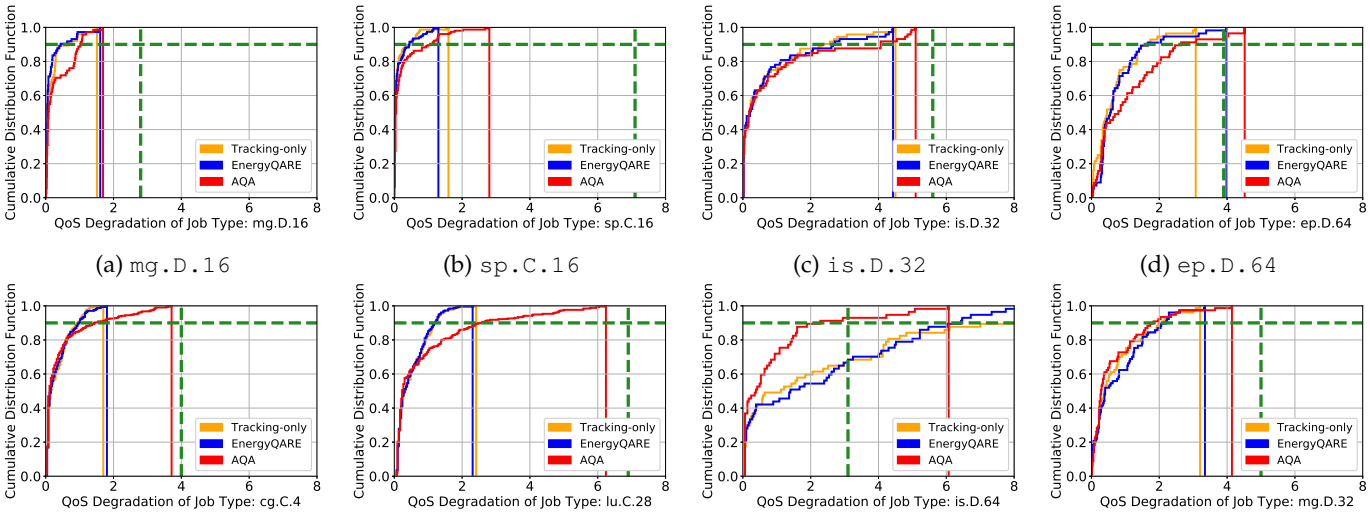


Fig. 14: Experiments on a real 36-server cluster running workload W4 using the EnergyQARE policy.



(a) `mg.D.16` (b) `sp.C.16` (c) `is.D.32` (d) `ep.D.64`

(e) `cg.C.4` (f) `lu.C.28` (g) `is.D.64` (h) `mg.D.32`

Fig. 15: The cumulative distribution functions (CDF) of all 8 applications' QoS degradation when running workload W4 in real-system experiments with three different policies. Both the EnergyQARE and the Tracking-only policies cannot meet the QoS constraints of application `is.D.64`, as shown in (g). On the other hand, our AQA policy can meet that application's QoS constraint by giving it a large weight as shown in Fig. 8. In these CDF curves, the solid vertical line shows where the curves reach 100%.