This paper was presented as part of the main technical program at IEEE INFOCOM 2011

# Managing Cost, Performance, and Reliability Tradeoffs for Energy-Aware Server Provisioning

Brian Guenter, Navendu Jain, and Charles Williams
Microsoft Research, Redmond, WA

***Abstract*—We present ACES, an automated server provisioning system that aims to meet workload demand while minimizing energy consumption in data centers. To perform energy-aware server provisioning, ACES faces three key tradeoffs between cost, performance, and reliability: (1) maximizing energy savings vs. minimizing unmet load demand, (2) managing low power draw vs. high transition latencies for multiple power management schemes, and (3) balancing energy savings vs. reliability costs of server components due to on-off cycles. To address these challenges, ACES (1) predicts demand in the near future to turn on servers gradually before they are needed and avoids turning on unnecessary servers to cope with transient load spikes, (2) formulates an optimization problem that minimizes a linear combination of unmet demand and total energy and reliability costs, and uses the program structure to solve the problem efficiently in practice, and (3) constructs an execution plan based on the optimization decisions to transition servers between different power states and actuates them using system and load management interfaces. Our evaluation on three data center workloads shows that ACES's energy savings are close to the optimal and it delivers power proportionality while balancing the tradeoff between energy savings and reliability costs.**

## I. Introduction

As the demand on Internet services increases, the energy consumed by data centers, particularly the servers hosted therein, has been skyrocketing. In 2006, servers and data centers consumed 61 billion kWh at a cost of $4.5 billion with demand expected to double by 2011 [3]. Recent studies estimate that a 300W server consumes about $330 of energy cost per year [25]. Unfortunately, much of this power is wasted at low utilization: in typical data centers, servers run at 30% utilization where they consume at least 65% of the peak power draw [9]. This wastage is compounded by losses in power delivery, power supply inefficiencies, and cooling infrastructure costs.

In response, computer vendors are developing power management techniques for computing [1], storage [14, 27], and networking equipment [5, 19]. These techniques aim to provide energy-proportional computing [6, 23] by (a) using consolidation to move workloads off under-utilized servers and then switching them to low power modes, and (b) adapting the rate of operation (e.g., CPU frequency) of active servers to offered load. However, three challenges confront operators who want to manage these techniques to save energy:

- **Maximizing energy savings vs. minimizing unmet load demand:** A simple way of resource provisioning is to keep all servers in the highest performance state. This approach ensures that the system meets the load demand under specified Service-Level Objectives (SLO) but consumes excessive energy under low load. A different approach is to operate all servers at a low enough voltage and frequency just to meet demand, but this risks SLO violations. Static or improper provisioning can result in performance loss and wasted energy under dynamic workloads, which are typical in data centers [8, 9, 12, 17].
- **Managing multiple power states:** Servers today support multiple power management schemes such as DVFS (ACPI P states), sleep or suspend-to-RAM (ACPI state S3) and hibernate (ACPI state S4). While these states offer fine-grained control between power and performance, they pose software complexity in managing power draw vs. transition latency tradeoffs—low power modes minimize power draw of idle servers, but incur a high delay to boot up the machine e.g., a server in hibernate mode draws 2W-5W compared to 200W when on, but it takes 30s-60s to transition into and out of that state.
- **Reducing the impact on reliability:** Repeated on-off cycles increase the wear-and-tear of server components (e.g., disks, fans), incurring costs for their procurement and replacement. Similarly, the processor lifetime can be affected by thermal effects due to on-off cycles. Our conversations with operators indicate that a large fraction of hardware induced failures are recurrent: a server crashing due to faulty hardware is highly likely to crash again. Hence, we need to balance the tradeoff between energy savings and reliability costs of repairing faulty server components.

Thus, the key question for server provisioning is: *when and how many servers to transition to each power state to meet demand while minimizing the energy and reliability costs?*

Unfortunately, prior approaches do not address all of these challenges. Many solutions [11, 12, 22] leverage dynamic voltage and frequency scaling (DVFS) where processor frequency and voltage can be controlled to save CPU power consumption. Although DVFS can be effective in reducing CPU power, processors account for only 25% power in servers today while other components still incur many fixed power overheads when active [17]. Some studies [8, 10, 12, 15, 20] use on-off primitives to temporarily shutdown a subset of servers during periods of low demand. However, on-off based techniques do not consider reliability costs, and they are typically *reactive* resulting in high latency to transition to active state, which risks performance loss under load spikes [11, 12, 20].

We design, implement, and evaluate ACES, an Automated

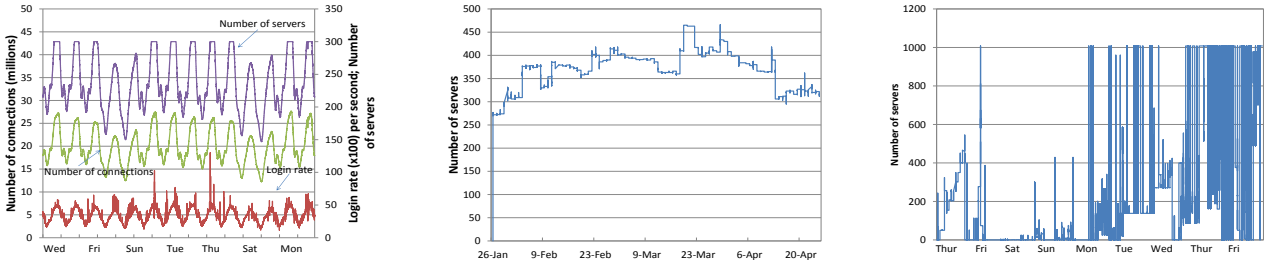


**Fig. 1:** Load patterns for the (a) Messenger, (b) Azure, and (c) SCC traces. We observe highly periodic nature of Messenger, small load variation for Azure, and a high dynamic range and sharp transient spikes for the SCC trace.

Controller for Energy-aware Server provisioning that provisions servers to meet workload demand while minimizing the energy cost and the maintenance and reliability cost due to duty cycling. To balance the tradeoff between unmet load demand and energy savings, ACES manages server power states to keep sufficient capacity readily available to cope with sudden load increase and to maximize energy savings during the idle intervals.

ACES integrates load prediction, system and load management functions, and cost-benefit analysis in three steps. First, ACES characterizes the input workload and uses regression analysis to predict workload demand in the near future. We validated the accuracy of our prediction algorithm against three workload traces from data centers. Second, ACES models server power schemes as a Markov state diagram where each state denotes a server power scheme and edges denote the transition delay and the dollar cost of energy usage and server maintenance. Using this model, we formulate the energy-aware server provisioning problem as an integer linear programming problem whose goal is to meet the predicted demand within delay constraints. We approximate the problem through a linear programming formulation and solve it with an efficient fast polynomial-time method [7]. Third, ACES uses the optimization decisions to determine the number of servers to keep in each power scheme and actuates these decisions using system management interfaces, for example, by using load balancers to remove load from servers transitioning from active to low-power states.

We have evaluated ACES on three workloads: Windows Azure [26], Live Messenger, and a computing cluster of about 1000 servers. Our results show that ACES adapts to offered load and provides energy savings close to the offline-optimal algorithm, for the workloads we considered.

This paper makes the following contributions.

- We analyze workload traces from production servers and data centers to determine demand variation and power profiles of servers. We find that servers are over 60% idle, demand is mostly stable and predictable, but spikes can be abrupt and the peak workload is 1.7x-6x higher than the average workload. Therefore, keeping all servers active wastes energy, yet the likelihood of failing to meet a sudden load spike must be minimized.
- We present a new automated server provisioning system that minimizes unmet workload demand while balancing the tradeoff between energy savings and reliability impact due to on-off cycles.
- Our analysis of three workload traces demonstrates that compared to the optimal, ACES saves energy within 96% while a greedy algorithm saves energy within 84%.

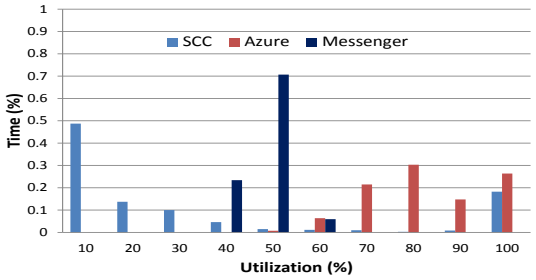


**Fig. 3:** Cluster utilization histogram for the three traces. For the SCC trace, the fraction of active servers is below 30% half of the time but is 100% utilized about 20% of the time. The Messenger and Azure clusters have about 50% utilization and about 80% utilization 70% of the time, respectively, with high load variation.

## II. Measurement Study and Related Work

### A. Analysis of Workload Traces

We begin by analyzing traces collected from two large data center applications, Windows Live Messenger and Windows Azure [26], and a shared computing cluster (SCC). Live Messenger is an instant messaging application and Azure is a cloud platform that provides services for development, hosting, and management of cloud applications. SCC is a distributed, parallel computation platform that runs large-scale scientific and computational research applications. The Live Messenger trace contains login rate and number of connections per server which we scaled down to 25M connections across 300 servers over two weeks; a prior study presented many aspects of load and user behavior for this system [9]. The Azure and SCC traces contain cluster utilization data collected from about 500 servers over three months and from 1008 servers over a week, respectively.

We analyze load patterns, auto-correlations, and cluster utilization for these traces. Figure 1(a) shows a pattern of

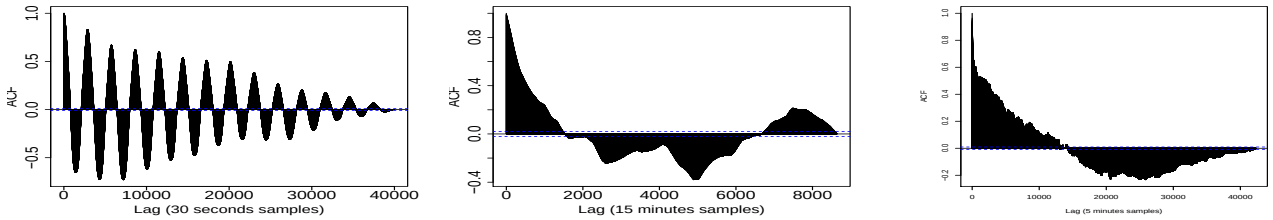


**Fig. 2:** Workload auto-correlation (ACF) for the (a) Messenger, (b) Azure, and (c) SCC traces.

the total number of connections and the login rates collected every 30s from the Messenger trace. We observe that the load exhibits diurnal patterns with the peak load being 2.5x-6x higher than the average load and that the login rate is noisier (i.e., transient load spikes) than the connection count. The top curve shows the corresponding load demand in terms of number of servers calculated using a simple provisioning model [9]. Figure 1(b) shows the server demand collected every 15 minutes for an Azure cluster. The peak load is about 1.7x compared to the minimum load. One reason for the stable load pattern is that the Azure pricing model that incentivizes long-term usage. Figure 1(c) shows job requests for the SCC trace. We observe significant fluctuation in the server demand from about 10% on the weekends to 100% during the weekdays corresponding to user job submissions.

Since transitioning servers to the active state takes time, load prediction can help us act early depending on the temporal correlation of load patterns in the workload. Note that the prediction time intervals can range from tens of seconds to several minutes to perform proactive provisioning for different workloads. Figure 2 shows the auto-correlation for these workloads. We find a strong daily and weekly autocorrelation for Messenger and a slightly elevated autocorrelation for Azure. However, the SCC trace has a high variation which would require short-term prediction scheme to adapt quickly to dynamic load. Overall, we observe highly periodic nature of Messenger, stable load patterns for Azure, and a high dynamic range and sharp transient spikes for the SCC trace.

Figure 3 depicts a histogram of cluster utilization in terms of number of servers for the three traces. The most significant feature of the SCC data is that the fraction of active servers is below 30% half of the time but is also 100% utilized about 20% of the time. The Messenger and Azure clusters have about 50% utilization and about 80% utilization 70% of the time, respectively, with high load variation.

We make two key observations from these results. First, the total load exhibits significant fluctuation over time, which provides the opportunity to dynamically change the number of active servers. Further, since an active server consumes about 60% of its peak power when idle, turning off servers during off-peak periods can maximize energy savings. Second, we want to leverage strong correlations in load patterns to predict future demand and perform proactive server provisioning to mask the high latency of transitions between different power states. Thus, to cope with load variations in dynamic workloads, we focus on designing optimization-based algorithms than ad hoc heuristics, as also advocated by others [16].

**Related Work.** There is a large body of work on dynamic server provisioning and energy management in data centers. Pinheiro et al. presented a simple scheme to power servers on and off dynamically [20]. Muse uses an economic approach to allocate servers, where services bid for resources as a function of required performance, and the system continuously monitors load and allocates resources based on its utility [8]. Raghavendra et al. propose a power management approach that coordinates different individual approaches to simultaneously minimize peak and average power, and idle power wastage [21]. Chen et al. use forecast-based and hysteresis-based techniques for server provisioning and combine it with load batching for connection-oriented services [9].

Other work based on control theory aims to dynamically optimize for energy, resources, and operational costs while meeting performance-based SLOs [10, 11]. Albers et al. provide a good survey on energy-efficient algorithms from a theoretical perspective [4]. Gandhi et al. propose combining dynamic voltage scaling and DVFS techniques to optimally allocate power in server farms [13].

We differ from prior work in that we integrate forecast-based provisioning with an optimization-based approach to maximize energy savings, manage multiple power states to meet demand while masking transition latencies between power states, and take into account the reliability costs due to duty-cycling.

## III. Design Overview

### A. System Model

Our hosting site model comprises a large number of identical servers in a data center, which can be provisioned to host applications, possibly in a virtualization environment. For hosted applications, it is important to have sufficient number of active servers to meet demand at a desired level of SLO performance. In particular, we analyze the client request rate a server can process under a bounded response time (or throughput rate) and use this metric to provision sufficient number of servers to handle the incoming load.

A server kept active to process workload consumes power, incurring electricity costs as well as power distribution (con-

version losses) and cooling costs to ensure stable operation. Therefore, the goal of the data center operator is to meet demand under specified SLO while minimizing the operational energy costs. While server consolidation can increase the overall utilization and decrease idle power by reducing the number of active servers, data centers still require sufficient capacity to handle the peak load. Further, consolidation does not save energy automatically as administrators need to actively consolidate services and turn off unneeded capacity. Finally, service robustness concerns often preclude consolidation of mission-critical services.

As discussed in Section I, there are two common mechanisms to manage server power consumption. First, most processors support DVFS, which reduces CPU power demand under light load. While prior studies [12, 22] show that DVFS offers significant CPU power savings, processors are not the dominant power consumer in servers today and controlling DVFS remains an active topic of research [17]. Further, our analysis in Section II shows that there are several periods of low cluster utilization during which the whole server can be switched off to save more energy. Therefore, we focus on using low power states (off, sleep, hibernate) as they yield high energy savings during periods of light load. While prior work [17, 22] does not leverage sleep and hibernate states for interactive services due to their high transition latency, we proactively provision to effectively mask their transition delays without reducing server availability. Note that delays due to software (e.g., migrating load off servers) can be simply added to the latency in transition between power states of the underlying hardware.

### B. Algorithm Overview

We propose ACES, an energy-conservation approach where servers transition between a high-performance state and low-power states in response to load. Our ACES design has two key components: a load predictor for proactive provisioning and an optimizer framework that uses workload prediction to manage transitions into and out of power states to meet demand while saving energy and reducing the reliability cost.

First, we employ forecast-based technique based on regression analysis to predict load in the near future. We quantize time into discrete time steps, which may be a few seconds to several minutes, depending on the load variation and the transition delays associated with changing server power states.

Second, to optimize state transitions to meet demand while saving energy, we build a Markov state diagram of power state transition for each server by measuring the power draw in each power state and the transition latencies in and out of these states. Figure 4 shows the state diagram for a server with on-off and on-off-sleep-hibernate power states. The circles denote the power states and the edges denote the cost of state transitions. The cost is expressed in terms of latency (time steps) and the dollar cost of energy usage and the reliability cost due to the transition. We have developed an analytical model (Section III-C) based on measurements to amortize reliability costs per power state transition. For ease of illustration, we focus on the on and off states and later generalize our approach to multiple power states.

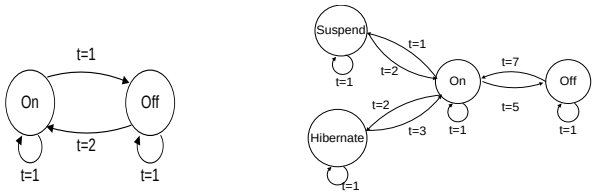


**Fig. 4:** Markov state diagram of transitions for (a) On-Off and (b) On-Off-Suspend-Hibernate power schemes. In the On-Off state diagram, the system can transition from On to Off in one time step and from Off to On in two time steps.

We use the Markov state representation of a single server to develop a state transition graph to manage power transitions across all servers. At each time step, the predictor estimates future machine demand for subsequent time steps $1, \ldots, k$. Note that the minimum time step interval should be equal to the highest common factor (HCF) of the state transition delays. The parameter $k$ is chosen such that the prediction interval is at least the maximum transition delay between any two power states. The state transition optimizer uses the estimated load to choose a set of power transitions at each time step such that the predicted machine demand is met as closely as possible, and the total energy and reliability cost is minimized. The power transitions that begin in the current time step are executed and then the entire process repeats. Note that power state transitions cannot be terminated while they are in progress.

### C. Quantifying the Reliability Impact

A key aspect to consider is the impact of transitioning servers between different power states, as it may significantly affect the long-term reliability of the system. Past studies have noted that many server components (e.g., disk drives, fans) can be susceptible to on-off cycles [10, 17, 22], thus increasing the failure rate when servers are turned off. Similarly, processor and motherboard reliability can be affected by temperature gradient and thermal stress, respectively. Note that the failure behavior of a component may also depend on the operating conditions e.g., temperature, humidity, and vibration, which are not considered in our model.

We use a simple analytical model based on [10] to quantify the reliability impact in terms of its dollar cost. We compute the amortized cost of a single power state transition as follows: First, we divide the total cost (procurement and replacement) per component by its MTTF (duty cycles), available via datasheets or empirical analysis e.g., a disk drive with datasheet MTTF of 50k boot cycles and cost $200 will incur a reliability cost of 0.4 cents per cycle. Second, we calculate the sum of per-cycle costs across all components; the amortized maintenance costs (e.g., to replace faulty components) can be similarly added to the model. We annotate each edge in the state machine diagram with the reliability cost per power state transition. Though this model is simple, we believe that it provides a reasonable approximation to model reliability costs.

| | |
|---|---|
| $X_{s_i u_j}$ | Number of servers transitioning from power state $s$ at time $i$ to power state $u$ at time $j$ |
| $P_{s_i u_j}$ | Energy required to make the transition $s_i$ to $u_j$ |
| $m_j$ | Server demand at time step $j$ |
| $On_j$ | Number of servers kept on at time step $j$ |
| $e_j$ | Absolute value of the difference between server demand and the number of servers on at time step $j$ |
| $E$ | Total error between server demand and the number of on machines over all time steps $j$ |
| $C$ | Total energy and reliability cost over all time steps $j$ |
| $\mathbf{a_j}$ | Predictor for time step $j$ |
| $\hat{\mathbf{A}}$ | Square sub matrix of equality constraint matrix $\mathbf{A}$ |

**Fig. 5:** Notation table.

## IV. Algorithm Design and Implementation

In this section we first present our prediction scheme to estimate load and then describe our optimization-based algorithm for server provisioning.

### A. Prediction-based server provisioning

ACES employs a load prediction algorithm to turn on servers gradually before they are needed and to keep adequate number of servers available to cope with sudden load spikes. Since the lead time from starting a server to getting it ready and moving from active to a low power state (e.g., ACPI S4) is significant, the prediction scheme needs to conservatively forecast the anticipated load change to minimize unmet demand and to maximize energy savings.

For many Internet services, the workload demand has periodic components in days, weeks, and months, and possibly a long term growth trend. For server provisioning, however, it suffices to focus on short-term load prediction. Several prediction schemes have been proposed on short-term load forecasting for seasonal time series [9]. We derive a simple and intuitive linear prediction scheme [24] because it is easy to implement, has a sound theoretical basis, and works extremely well on our workload traces. Nonetheless, the following discussion applies no matter which prediction scheme is used.

In linear prediction, the future demand $\hat{m}_0$ is predicted by taking the dot product of coefficient vector $\mathbf{a}$ with a vector of past demand values, $m_i$ $(i < 0)$,

$$\hat{m}_0 = [a_1 \dots a_p] \begin{bmatrix} m_{-p} \\ \vdots \\ m_{-1} \end{bmatrix} \tag{1}$$

where $(a_1, \dots, a_p)$ denote the $p$ coefficients of a weighted linear predictor and are computed by solving the least-squares problem $\mathbf{Ma} = \mathbf{m}$ i.e.,

$$\begin{bmatrix} w_{-r} & & 0 \\ & \ddots & \\ 0 & & w_{-1} \end{bmatrix} \begin{bmatrix} m_{-(r+p)} & \dots & m_{-(r-1)} \\ & \vdots & \\ m_{-p} & \dots & m_{-1} \end{bmatrix} \mathbf{a} = \begin{bmatrix} w_{-r} & & 0 \\ 0 & \ddots & \\ & & w_{-1} \end{bmatrix} \begin{bmatrix} m_{-r} \\ \vdots \\ m_0 \end{bmatrix} \tag{2}$$

where $w_{-i} = \alpha^{i-1}$; $\alpha = 0.95$ in our implementation.
Note that the number of coefficients $p$ and the number of past samples $r$ are both chosen large enough to capture the sufficiently long-term trends, with $r \gg p$ to prevent overfitting the predictor to a small sample of past values.

Since we estimate load for $k$ time steps in the future, we use $k$ predictors, $\mathbf{a_1} \dots \mathbf{a_k}$. The coefficients of each of the $k$ predictors are updated at every time step, which requires solving $k$ least squares problems. For typical values of $p, k$, and $r$ (typically $p \leq 10, k \leq 20$ and $r \leq 100$), we employ the singular value decomposition (SVD) method to solve the least-squares equations [18] i.e., $\text{SVD}(\mathbf{M}) = \mathbf{U}\sigma\mathbf{V^T}$ where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices and $\sigma$ is a diagonal matrix with the singular values along the diagonal. To be robust to noise, we discard small singular values $\sigma_i$

$$\sigma_{\mathbf{i}}^{'} = \begin{cases} \sigma_i & \text{if } \frac{\sigma_0}{\sigma_i} \leq s_{max} \\ 0 & \text{otherwise} \end{cases}$$

before inverting $\mathbf{U}\sigma\mathbf{V^T}$ to solve for $\mathbf{a}$; we use $s_{max}$=100.

The SVD has the advantage of numerical robustness: accurate results will be computed regardless of the rank of the data matrix $\mathbf{M}$ and the effective size of the linear predictor automatically adapts to the data, assuming $p$ is sufficiently large. Further, computing the SVD for the maximum values of $p$, $k$, and $r$ given above takes less than 300ms using a single core of a 2.4 GHz Core 2 Duo processor. For larger values of $p$, $k$, and $r$, asymptotically faster recursive methods [24] can be used, but care must be taken to avoid numerical instability.

**Evaluating prediction accuracy.** We evaluate the accuracy of our weighted linear prediction scheme by comparing the predicted values measured every 30s, 15 mins, and 5 mins, against the actual demand for the number of servers in the Messenger, Azure, and SCC traces, respectively [2]. Using this model, we can forecast the smooth trends of Messenger and Azure traces with good accuracy. The sharp load spikes are hard to predict, because they are caused by server crashes that result in re-login bursts for Messenger and by bursty job arrivals in case of SCC.

We compute the standard deviation $\sigma$ of the relative error $(|\text{Predicted} - \text{Actual}|/\text{Actual})$ as goodness of fit metric. We find that the standard deviation is 0.009, 0.014, and 0.073, respectively, for the Messenger, Azure and SCC workloads.

Note that the time series can be noisy. One approach would be to use a longer forecasting interval or do a coarse-grained prediction, but that may risk long periods of overprovisioning or underprovisioning. Instead, we predict the load at multiple time-steps and expect load prediction to work reasonably well for intervals of tens of seconds to a few minutes.

### B. Machine Transition Graph

The first step in formulating the optimization problem is to unroll in time the Markov state diagram for a single server. Figure 6 shows the machine transition graph for Figure 4. In this graph a vertex $s_j$ represents a power state $s$ at a given time step $j$ and an edge represents the transition of servers from one power state to another. Each edge has a value $X_{s_i u_j}$ denoting the number of servers that transition from power state $s$ at time $i$ to power state $u$ at time $j$ (see Figure 6(b)).

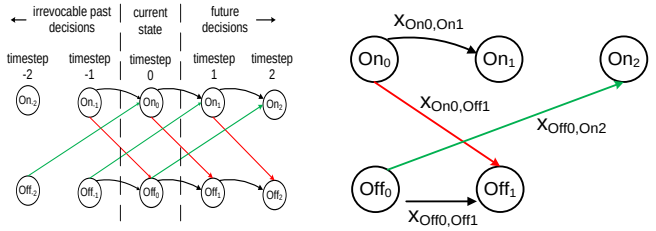


**Fig. 6:** (a) State diagram unrolled in time. If the timestamp of the second vertex of an edge is less than zero it cannot affect the current state or future decisions; these edges are not shown. For the On-Off system no states occurring earlier than time step -2 can affect the current state; these states are not shown. (b) Edge labeling convention.

Assuming that the number of servers in the system is constant with time (we weaken this assumption in Section IV-D), we can write the conservation equation in terms of the number of servers as:

$$\forall s, j : \sum_{k>j} X_{s_j u_k} - \sum_{i<j} X_{r_i s_j} = 0 \tag{3}$$

which denotes that the total number of servers entering power state $s$ at time $j$ must equal the total number of servers leaving power state $s$ after time $j$; note that the servers remaining in a given power state transition to the *same* state at time $j+1$.

There are two types of edges in the machine transition graph. The edges $X_{s_i on_j}$ with $i < 0$ represent power transition decisions in the past and cannot be changed e.g., an edge $X_{off_{-1} on_2} = 10$ represents a decision, made one time step in the past, to transition 10 servers from the off state to the on state two time steps in the future. These servers have already begun the boot cycle and cannot be controlled again till after they have reached the on state in time step 2. The other type of edges, $X_{s_i on_j}$, with $i \geq 0$, represent a server transition in the future; these edges are the variables to be solved for. Separating out the constants and variables, we get the following modified conservation equations:

$$\overbrace{\sum_{k>j} X_{s_j u_k} - \sum_{i\geq 0} X_{r_i s_j}}^{\text{variables}} = \overbrace{\sum_{i<0} X_{r_i s_j}}^{\text{past}} = b_{s_j} \tag{4}$$

There is one such equation for each node in the transition graph which gives us a set of conservation equations: $\mathbf{Ax} = \mathbf{b}$.

**Tracking Previous Decisions.** Incorporating the results of previous server transition decisions into the current optimization step is easily accomplished by moving edge values back one step in time for every prediction value.

Figure 7 illustrates a sequence of prediction steps for optimizing state transitions. In the first prediction step, 11 machines are on and 10 are off at time step zero. Based on predicted future demand, the optimizer computes machine transitions between time steps 0 and 1, and between time steps 1 and 2. Machine transitions for time step zero are executed, the machine transition values are moved backward one step in time, and the system advances one prediction step. The entire process is repeated for the second and third prediction steps.

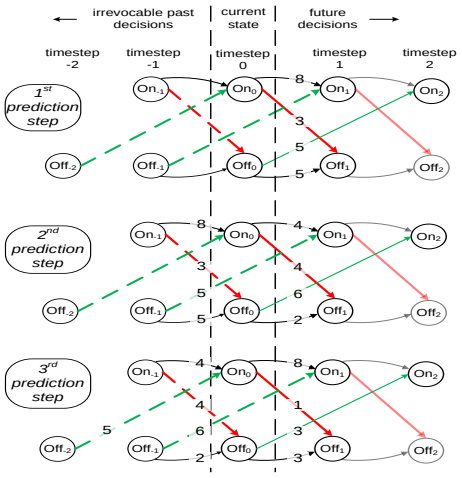


**Fig. 7: Moving edge values back in time.**

### C. State Transition Optimization

Our goal is to meet the machine demand $m_j$ at every time step while minimizing the costs of energy usage and reliability impact. Therefore, the objective function we want to minimize has two components. First, the difference $E$ between predicted server demand $m_j$ at time step $j$ and the number of active servers $On_j = \sum_i x_{s_i on_j}$ over all time steps.

$$E = \sum_j |m_j - On_j| = \sum_j \left| m_j - \sum_i x_{s_i on_j} \right| = \sum_j e_j \tag{5}$$

The second component is the total cost $C$ computed as the sum over all edges having cost $P_{s_i u_j}$ and denoting transitions between power states, times the number of machines $X_{s_i u_j}$ making the transition.

$$C = \sum_{i\geq 0} P_{s_i u_j} X_{s_i u_j} \tag{6}$$

Given these costs, we formulate the optimization problem as an integer linear program (ILP) of first minimizing the unmet server demand $E$ and then the transition cost $C$ i.e.,

$$\text{MIN } E \text{ then } C \tag{7}$$

$$\text{s.t. } \mathbf{Ax} = \mathbf{b} \tag{8}$$

$$e_j \geq m_j - On_j \tag{9}$$

$$e_j \geq -(m_j - On_j) \tag{10}$$

For some applications, server demand and power consumption may be interchangeable, i.e., it is acceptable to have small unmet demand if enough power can be saved. In this case, we can formulate the objective function as MIN $\gamma * E + (1-\gamma) * C$, where $\gamma \in [0, 1]$ is a weighing constant.

**Integrality of the solution.** In general ILP problems can take exponential time in the worst-case. We make a key observation about our problem structure that even if the constraint matrix for this problem is not totally unimodular, the basic feasible solutions are guaranteed to be integer. This property allows the

ILP problem to be solved efficiently using linear programming in polynomial time; details appear in a technical report [2].

**Actuation of server provisioning.** After computing the number of servers required to process demand under application-specific SLO, our system makes two actuations. First, it uses the power state transitions primitives to switch servers between different power states to meet load demand while saving power. Second, it adjusts load dispatching to distribute load only among the remaining active servers. Further, we combine load dispatching with server provisioning to intentionally skew load on servers (instead of load balancing) so as to create *tail* servers with relatively low load such that turning them to low-power states will not affect the application performance [9].

### D. Discussion

**Heterogeneous servers.** Enterprise and data center environments often host servers with heterogeneous configurations due to application-specific requirements, equipment upgrades, legacy systems, etc. To support server heterogeneity, we model servers with different types, $G$ , with each type governed by a different state machine diagram. Our goal is to meet server demand $D_g$ for each type $g \in G$ while minimizing the energy and reliability costs over predicted demand at time steps $1, \ldots, k$. For servers in each server type, we build their machine transition graph and write the conservation equations 4 as follows:

$$\begin{bmatrix} \mathbf{A}_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathbf{A}_{|G|} \end{bmatrix} \mathbf{X} = \mathbf{b} \tag{11}$$

where $\mathbf{X}$ is now a vector containing all the edges in all $|G|$ graphs. We compute the unmet demand $E_G$ over all $|G|$ graphs as the accumulated difference between demand $m_j^g$ of type $g$ and the number of active servers $On_j^g = \sum_i X_{s_i on_j}^g$ i.e.,

$$E_G = \sum_{g=1}^{|G|} \sum_j \left| m_j^g - \alpha_g * On_j^g \right| = \sum_j e_j^g \tag{12}$$

where $X_{s_i on_j}^g$ is an edge value taken from machine transition graph $g$, and $\alpha_g$ is a weight which represents the relative importance of meeting server demand for type $g$. The total cost $C_G$ over all $|G|$ graphs becomes:

$$C_G = \sum_{g=1}^{|G|} \sum_{i \geq 0} P_{s_i u_j}^g \, X_{s_i u_j}^g \tag{13}$$

and run the state optimization algorithm of section IV-C using equations (7), (11), (12), and (13).

**Dynamic server availability.** In practice, the number of available servers can vary over time due to failures, upgrades, addition of new servers, etc. Our algorithm can be easily extended to accommodate dynamic server membership: if the number of machines that enter a power state at a given time is not equal to the number that were commanded to enter that state then the edge value can simply be updated to reflect the current, true situation. The state transition optimization can then be run with the updated edge values. Further, operators may always want to keep an additional buffer capacity (e.g., 10% servers) over the predicted demand to handle failures.

**Selecting servers for power state transitions.** There are several criteria to consider when deciding which servers to power on or off. The decision can be based on a combination of factors such as load, trying to equalize power-on cycles across all servers (e.g., using round-robin scheduling based on keeping count of on-off cycles per server), preferentially powering off servers in hot spots or those that require service or upgrades, or based on server hardware and reliability characteristics.

## V. Experimental Evaluation

In this section we compare the performance of our server provisioning algorithm to the offline optimal algorithm and a greedy algorithm, based on three data center workloads described in Section II-A.

**Alternative server-provisioning algorithms.** The offline optimal algorithm executes the state transition optimization as in our approach, but it has the complete knowledge of the server demand at all future time steps. The greedy heuristic we consider is to order power states by decreasing power footprint and increasing latency to transition from that state to the active state. If the load prediction results in capacity increase, then transition the required number of servers to active in sequence of ordered power states. On decrease, transition the excess number of servers evenly among the low-power states. This heuristic represents a reasonable approach to optimize energy savings and, indeed, performs fairly well.

### A. Methodology

For evaluating these algorithms, we replay the workload traces to a centralized machine emulating the cloud setup with knowledge of the request (or job) demand and the current cluster utilization. To calculate energy savings, we assume that when we turn on a server, it consumes power immediately but delivers no capacity until the boot-up latency. This assumption may overestimate the unmet demand since it is likely that a server may be available in a shorter time. Similarly, when a server is commanded to a low power mode, it consumes power of the active state during the transition but provides no capacity. This simplification is conservative since the server may shutdown faster or may consume less energy over the transition period. Given that CPUs only account for 25% power in servers today [17], we do not consider the variation in the power usage of a server at different CPU utilizations.

The total power usage of the system is computed as the sum of the power drawn by a server in a given power state times the number of active servers in that power state, across all power states. We compare performance along the metrics of energy savings (with respect to keeping all servers in the highest active state), power proportionality, and the reliability impact.

We assume the following configuration based on server power measurements: each server consumes 200W, 11W, 2W,

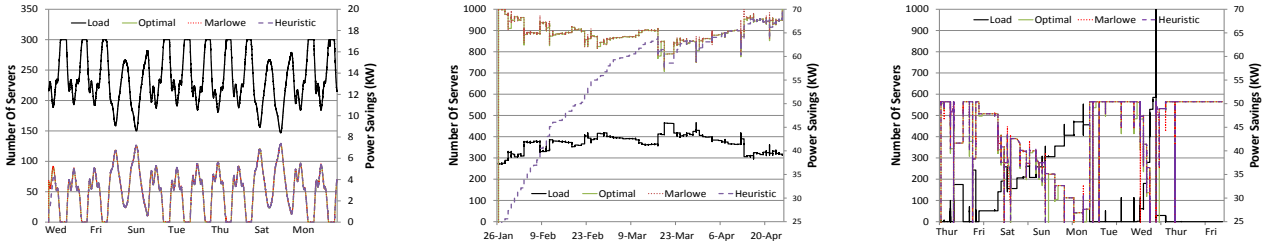


**Fig. 8:** Input demand (number of servers) and the power consumed by the three algorithms for (a)Messenger, (b)Azure, and (c)SCC traces.

and zero watts in on, sleep, hibernate, and off power states, respectively. For SCC, the on to off, off to on, on to sleep, sleep to on, on to hibernate, and hibernate to on transition latencies are 45s, 45s, 5s, 15s, 30s, and 60s, respectively. For Azure, the corresponding latencies are 60s, 1200s, 5s, 1200s, 30s, and 1200s, respectively. For Messenger, the corresponding latencies are 2 hours, 90s, 2 hours, 15s, 2 hours, and 60s, respectively. Since Messenger is a stateful service, we use a high delay (2 hours) to transition from active to a low power state in order to emulate the natural departure rate caused by normal user logoffs [9]. Using our analytical reliability model, market cost of server components and MTTF (based on conversations with data center operators), we calculate the amortized reliability cost due to power state transitions as 6 cents per transition from active to a low-power state.

### B. Evaluating energy savings

The first experiment evaluates the savings from energy-aware server provisioning for the three workloads. Figure 8 shows the workload demand in terms of number of servers and the power consumed by optimal, Marlowe, and greedy algorithms. For all three traces, we observe that the power curve usage follows variation in the workload. Since the Messenger trace exhibits a predictable load pattern, both Marlowe and greedy approaches consume power closer to the optimal. For the Azure trace, however, the load variation results in the greedy approach incurring significant delay in meeting the demand initially while Marlowe still performs close to the optimal. Finally, for the SCC trace, the performance gap between Marlowe and the greedy approach increases significantly as the high load variation results in excess capacity or high unmet demand using the greedy algorithm.

Figure 9 shows the bar graph denoting the normalized energy savings per day of ACES and the greedy algorithm compared to the offline optimal algorithm. We observe that compared to the optimal, ACES saves 96%-99.5% energy whereas the greedy approach saves 23%-84% energy for the three workload traces.

**Power proportionality.** Next, we compute the power proportionality of our algorithm. Figure 10 shows the corresponding results. The x-axis denotes the fraction of servers kept active by ACES and the y-axis denotes the fraction of energy consumed to total energy used when all machines are powered on. For the Messenger and Azure workloads, we observe that ACES consumes power proportional to load. For the SCC trace, the dynamic load patterns result in high variation in the power consumption, but overall, the total power usage is proportional to the input server demand.

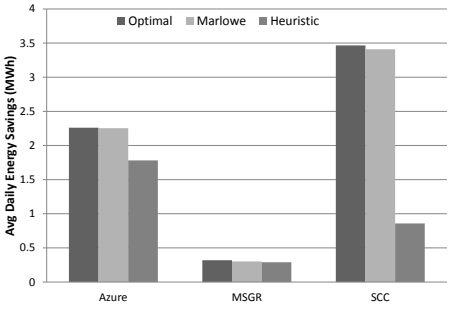


**Fig. 9:** Comparing normalized daily energy savings for ACES and greedy algorithms compared to the offline optimal.

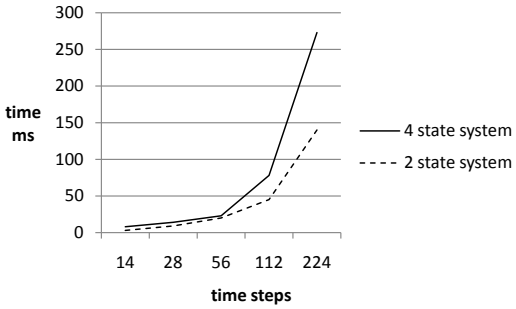


**Fig. 11:** LP solve time versus the number of time steps for the on-off power states and the on-off-sleep-hibernate power states.

### C. Micro-benchmarks

Finally, we evaluate the execution time to solve Equation (7) in computing optimal server state transitions in the state transition graph. In Figure 11, we plot the execution time on the y-axis and the number of lookahead timesteps on the x-axis. The two lines in the graph denote the number of server power states: two (on-off) and four (on-off-sleep-hibernate) corresponding to figure 4. We find that the time taken to solve (7) scales both with the size of the state diagram and with the size of the machine demand prediction window.

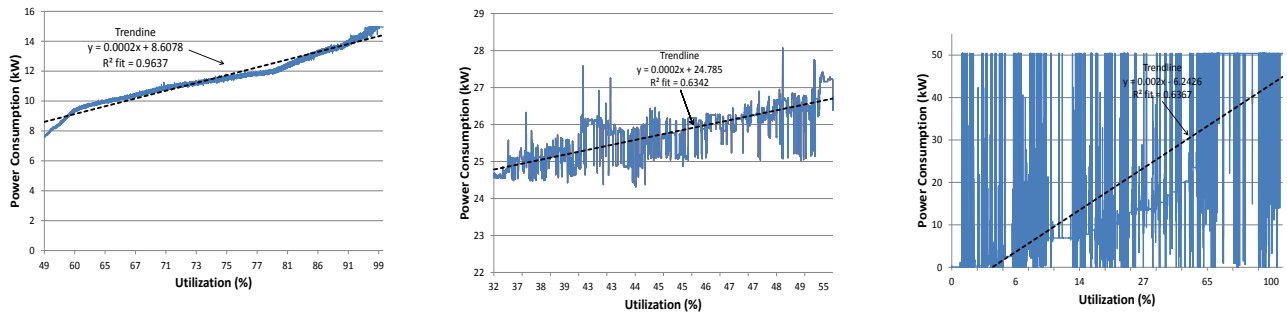


**Fig. 10:** Evaluating power proportionality of ACES for the (a) Messenger, (b) Azure, and (c) SCC traces.

As an example, if we assume each time step is 5 seconds, the longest prediction window would be approximately 20 minutes. Further, the computation time is scaling slightly less than quadratically with the size of the prediction window, and approximately linearly with the number of states; it is noteworthy that the execution time is independent of the number of servers in the system.

## VI. Conclusions and Future Work

> A dual strategy is needed to solve the energy problem: (1) maximize energy efficiency and decrease energy use; (2) develop new sources of clean energy. No. 1 will remain the lowest hanging fruit for the next few decades.
> *–Steve Chu, U.S. Secretary of Energy*

In this paper we presented ACES, an automated system for energy-aware server provisioning that minimizes unmet demand while reducing energy and reliability costs in hosting clusters. Our results show that the system can quickly identify the optimal assignment of servers to power schemes thereby significantly saving energy while meeting workload demand. While this paper focuses on server power management at the ensemble layer, we believe that new hardware designs with finer levels of power control for individual components such as disks, memory, and power supplies can be incorporated for both local and global energy efficiency.

Further research is needed to expand the generality of our approach. First, centralized server provisioning systems considered in this paper, typically scale to few hundreds of nodes and are limited to a single administrative domain. Therefore, hierarchical approaches are needed to scale to large systems comprising tens of thousands of servers. Second, for global energy management in data centers, coordinated management is needed between compute, storage, and network resources. Further, we need to coordinate server provisioning across computing, storage, and networking equipment as well as scheduled repair and maintenance tasks in data centers.

## References


[1] ACPI. Advanced Configuration and Power Interface Specification, Revision 3.0b. http://www.acpi.info.
[2] http://research.microsoft.com/en-us/um/people/navendu/marlowe-tr.pdf
[3] http://www.eia.doe.gov/cneaf/electricity/page/co2_report/co2report.html
[4] Albers, S.: Energy-efficient algorithms. IEEE Computer 53(5) (2010)
[5] Ananthanarayanan, G., Katz, R.H.: Greening the switch. In: HotPower (2008)
[6] Barroso, L.A., Hölzle, U.: The case for energy-proportional computing. IEEE Computer 40(12) (2007)
[7] Bertsimas, D., Tsitsiklis, J.N.: Introduction to Linear Optimization. Athena Scientific. ISBN-10: 1-886529-19-1
[8] Chase, J.S., Anderson, D.C., Thakar, P.N., Vahdat, A.M., Doyle, R.P.: Managing energy and server resources in hosting centers. SIGOPS Oper. Syst. Rev. 35(5), 103–116 (2001)
[9] Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., Zhao, F.: Energy-aware server provisioning and load dispatching for connection-intensive internet services. In: NSDI. pp. 337–350 (2008)
[10] Chen, Y., Das, A., Qin, W., Sivasubramaniam, A., Wang, Q., Gautam, N.: Managing server energy and operational costs in hosting centers. In: SIGMETRICS. pp. 303–314 (2005)
[11] Elnozahy, E.N., Kistler, M., Rajamony, R.: Energy conservation policies for web servers. In: USITS. pp. 8–8 (2003)
[12] Elnozahy, E.M., Kistler, M., Rajamony, R.: Energy-efficient server clusters. In: Proceedings of the 2nd Workshop on Power-Aware Computing Systems. pp. 179–196 (2002)
[13] Gandhi, A., Harchol-Balter, M., Das, R., Lefurgy, C.: Optimal power allocation in server farms. In: SIGMETRICS '09. pp. 157–168 (2009)
[14] Gurumurthi, S., Sivasubramaniam, A., Kandemir, M., Franke, H.: Drpm: dynamic speed control for power management in server class disks. SIGARCH Comput. Archit. News 31(2), 169–181 (2003)
[15] Heath, T., Diniz, B., Carrera, E.V., Jr., W.M., Bianchini, R.: Energy conservation in heterogeneous server clusters. In: PPOPP (2005)
[16] Keeton, K., Kelly, T., Merchant, A., Santos, C., Wiener, J., Zhu, X., Beyer, D.: Dont settle for less than the best: use optimization to make decisions (2007)
[17] Meisner, D., Gold, B.T., Wenisch, T.F.: Powernap: eliminating server idle power. In: ASPLOS (2009)
[18] Meyer, C.D.: Matrix Analysis and Applied Linear Algebra (2000)
[19] Nedevschi, S., Popa, L., Iannaccone, G., Ratnasamy, S., Wetherall, D.: Reducing network energy consumption via sleeping and rate-adaptation. In: NSDI. pp. 323–336 (2008)
[20] Pinheiro, E., Bianchini, R., Carrera, E.V., Heath, T.: Load balancing and unbalancing for power and performance in cluster-based systems. In: Workshop on Compilers and Operating Systems for Low Power (2001)
[21] Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., Zhu, X.: No "power" struggles: coordinated multi-level power management for the data center. In: ASPLOS (2008)
[22] Rao, L., Liu, X., Xie, L., Liu, W.: Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment. In: INFOCOM (2010)
[23] Tolia, N., Wang, Z., Marwah, M., Bash, C., Ranganathan, P., Zhu, X.: Delivering energy proportionality with non energy-proportional systems - optimizing the ensemble. In: HotPower (2008)
[24] Vaidyanathan, P.: The Theory of Linear Prediction. Morgan & Claypool. ISBN 9781598295764
[25] Wang, X., Chen, M.: Cluster-level feedback power control for performance optimization. In: HPCA. pp. 101 –110 (2008)
[26] Windows Azure Platform, http://www.azure.com
[27] Zhu, Q., Chen, Z., Tan, L., Zhou, Y., Keeton, K., Wilkes, J.: Hibernator: helping disk arrays sleep through the winter. In: SOSP (2005)