Energy Efficient Scheduling of Application Components via Brownout and Approximate Markov Decision Process

Minxian Xu(✉) and Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory,
School of Computing and Information Systems,
The University of Melbourne, Melbourne, Australia
minxianx@student.unimelb.edu.au, rbuyya@unimelb.edu.au

Abstract. Unexpected loads in Cloud data centers may trigger overloaded situation and performance degradation. To guarantee system performance, cloud computing environment is required to have the ability to handle overloads. The existing approaches, like Dynamic Voltage Frequency Scaling and VM consolidation, are effective in handling partial overloads, however, they cannot function when the whole data center is overloaded. Brownout has been proved to be a promising approach to relieve the overloads through deactivating application non-mandatory components or microservices temporarily. Moreover, brownout has been applied to reduce data center energy consumption. It shows that there are trade-offs between energy saving and discount offered to users (revenue loss) when one or more services are not provided temporarily. In this paper, we propose a brownout-based approximate Markov Decision Process approach to improve the aforementioned trade-offs. The results based on real trace demonstrate that our approach saves 20% energy consumption than VM consolidation approach. Compared with existing energy-efficient brownout approach, our approach reduces the discount amount given to users while saving similar energy consumption.

Keywords: Cloud energy efficiency · Application component · Microservices · Brownout · Markov decision process

1 Introduction

Given the scenario that budget and resource are limited, overloaded situation may lead to performance degradation and resource saturation, in which some requests cannot be allocated by providers. Thus, some users may experience high latencies, and others may even not receive services at all [14], which directly affects the requests that have Quality of Service (QoS) constraints. Unfortunately, current resource management approaches, like Dynamic Voltage Frequency Scaling (DVFS) [13] and VM consolidation [18], cannot function when the **holistic** data center is overloaded. The saturated resource not only brings over-utilized situation to hosts, but also causes high energy consumption.

Energy consumed by the cloud data centers has currently become one of the major concerns of the computing industry. It is reported that U.S. data centers will consume 140 billion kWh of electricity annually by 2020, which equals to the annual output of about 50 brown power plants [9]. Analysts also forecast that data centers will roughly triple the amount of electricity consumed in the next decade [2]. The servers hosted in data centers dissipate heat and need to be maintained in a fully air-conditioned and engineered environment. Though the cooling system is already efficient, servers remain one of the major energy consumers. One of the main reasons of high energy consumption lies in that computing resource are not efficiently utilized by server applications. Currently, building applications with microservices provides a more efficient approach to utilize infrastructure resource.

Applications can be constructed via set of self-contained components which are also called microservices. The components encapsulate its logic and expose its functionality through interfaces, which makes them flexible to be deployed and replaced. With components or microservices, developers and users can benefit from their technological heterogeneity, resilience, scalability, ease of deployment, organizational alignment, composability and optimization for replaceability [16]. This also brings the advantage of more fine-grained control over the application resource consumption.

Therefore, we take advantage of a paradigm called **brownout** [14] to handle with overloaded situation and save energy. It is inspired by the concept of brownout in electric grids and originates from the voltage shutdown that copes with emergency cases, in which light bulbs emit fewer lights and consume less power [10]. In Cloud scenario, brownout can be applied to applications components or microservices that are allowed to be disabled temporarily.

It is common that application components or microservices have this brownout feature. A brownout example for online shopping system is introduced in [14], in which the online shopping application provides a recommendation engine to recommend products that users may be interested in. The recommendation engine component helps service provider to increase profits, but it is not required to be running all the time. Recommendation engine also requires more resource in comparison with other components. Accordingly, with brownout, under overloaded situation, the recommendation engine could be deactivated to serve more clients who require essential services and have QoS constraints. Another example is the online document process application that contains the components for spell checking and report generation. These components are not essential to run all the time and can be deactivated for a while to reduce resource utilization. Apart from these two examples, brownout is available for other application components or microservices that are not required to be available all the time.

In this paper, we consider component-level control in our system model. The model could also be applied to container or microservices architecture. We model the application components as either mandatory or optional, and if required, optional components can be deactivated. By deactivating the optional components selectively and dynamically, the application utilization is reduced to save total energy consumption. While under market scenario, service provider may provide discount for users as one or more services are deactivated.

In our scenario, the meaning of discount is not limited to the discount offered to users. Additionally, it can also be modelled as the revenue loss of service providers (i.e. SaaS service providers) that they charge lower price for services under brownout. For example, in an online shopping system, the recommendation engine helps the service provider to improve their revenue by recommending similar products. If the recommendation engine is deactivated, the service provider is unable to obtain the revenue from recommendation engine.

The key **contributions** of this paper are: our approach considers the trade-offs between saved energy and the discount that is given to a user if components or microservices are deactivated; we propose an efficient algorithm based on brownout and approximate Markov Decision Process that considers the aforementioned trade-offs and achieves better trade-offs than baselines.

The remainder of this paper is organized as follows: after discussing the related work in Sect. 2, we present the brownout system model and problem statement in Sect. 3. Section 4 introduces our proposed brownout-based Markov Decision Process approach, and Sect. 5 demonstrates the experimental results of our proposed approach. The summary along with the future work are concluded in Sect. 6.

2 Related Work

A large body of literature has focused on reducing energy consumption in cloud data centers, and the dominant categories for solving this problem are VM consolidation and Dynamic Voltage Frequency Scaling (DVFS).

VM consolidation is viewed as an act of combining into an integral whole, which saves energy by allocating work among fewer machines and turning off unused machines [18]. Using this approach, VMs allocated to underutilized hosts are consolidated to other servers and the remaining hosts are transformed into low power mode. Mastroianni et al. [15] presented a self-organizing and adaptive approach for consolidation of VMs CPU and RAM resource, which is driven by probabilistic processes and local information. Corradi et al. [8] considered VM consolidation in a more practical viewpoint related to power, CPU and networking resource sharing and tested VM consolidation in OpenStack, which shows VM consolidation is a feasible solution to lessen energy consumption.

The DVFS technique introduces a trade-off between computing performance and energy consumed by the server. The DVFS technique lowers the frequency and voltage when the processor is lightly loaded, and utilizes maximum frequency and voltage when the processor is heavily loaded. Kim et al. [13] proposed several power-aware VM schemes based on DVFS for real-time services. Hanumaiah et al. [12] introduced a solution that considers DVFS, thread migration and active cooling to control the cores to maximize overall energy efficiency.

Most of the proposed brownout approaches in Cloud scenarios focused on handling overloads or overbooking rather than energy efficiency perspective. Klein et al. [14] firstly borrowed the approach of brownout and applied it to cloud applications, aiming to design more robust applications under unpredictable loads. Tomas et al. [19] used brownout along with overbooking to ensure graceful degradation during load spikes and avoid overload. In a brownout-compliant application or service, the optional parts are identified by developers, and a control knob called **dimmer** that controls these optional parts is also introduced. The dimmer value represents a certain probability given by a control variable and shows how often these optional parts are executed. Moreover, a brownout controller is also required to adjust the dimmer value.

Markov Decision Process (MDP) is a discrete time stochastic optimization approach and provides a way to solve the multiple state probabilistic decision-making problem, which has been adopted to solve resource management problems in Cloud scenarios. Toosi et al. [20] used finite MDP for requests admission control in Clouds, while their objective is maximizing revenues rather than reducing power consumption. Han et al. [11] applied MDP to determine VM migration for minimizing energy consumption, while our work is adopting MDP to determine the deactivation of application components.

In our previous work [21], several heuristic policies were proposed to find the components that should be deactivated and investigated the trade-offs between energy and discount. In this paper, we adopt approximate MDP to improve the aforementioned trade-offs.

3 System Model and Problem Definition

3.1 System Model

Our system model is presented in Fig. 1 and it consists of the following entities:

Users: Users submit service requests to cloud data centers. The users entity contains user information and requested applications (services).

Applications: The applications provide different services for users and are consisted of a set of components, which are identified as mandatory or optional.

Mandatory component: The mandatory component keeps running all the time when the application is launched.

Optional component: The optional component can be set as activated or deactivated according to the system status. These components have parameters like utilization $u(App_c)$ and discount $d(App_c)$. Utilization indicates the amount of utilization, and discount represents the amount of discount that is offered to the users (or revenue loss of service provider). The operations of optional components are controlled by the **brownout controller**, which makes decisions based on the system overloaded status and brownout algorithm.

To adapt the dimmer to our model, different from the dimmer in [14] that requires a dimmer per application, our dimmer is only applied to the applications with optional components. Rather than response time, another adaptation is that our dimmer value is computed based on the number of overloaded hosts and adapts to the severity of overloaded events (more details are presented in Sect. 4.1).

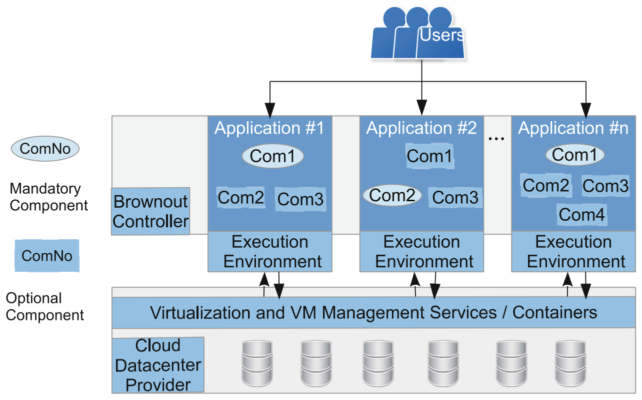

Fig. 1. System model with brownout

Cloud Providers: Cloud providers offer physical resources to meet service demands, which host a set of VMs or containers to run applications.

3.2 Power Model

We adopt the servers power model derived from [22]. The power of server i is $P_i(t)$ that is dominated by the CPU utilization:

$$P_i(t) = \begin{cases} P_i^{idle} + \sum_{j=1}^{N_i} u(VM_{i,j}(t)) \times P_i^{dynamic} & , N_i > 0 \\ 0 & , N_i = 0 \end{cases} \tag{1}$$

$P_i(t)$ is composed of idle power and dynamic power. The idle power is regarded as constant and the dynamic power is linear to the total CPU utilization of all the VMs on the server [22]. If no VM is hosted on a server, the server is turned off to save power. $VM_{i,j}$ refers to the jth VM on server i, N_i means the number of VMs assigned to server i. And $u(VM_{i,j}(t))$ refers to the VM utilization at time interval t, which is represented as:

$$u(VM_{i,j}(t)) = \sum_{c=1}^{C_j} u(App_c) \tag{2}$$

where C_j is the number of application components on VM, and $u(App_c)$ is the utilization of application component c when it is activated.

Then the total energy consumption during time interval t, with M servers is:

$$E(t) = \sum_{i=1}^{M} \int_{t-1}^{t} P_i(t)dt \tag{3}$$

Notes: In our power model, we assume that the time required to turn on/off hosts (including the time to deactivate and activate components) is lees than a scheduling time interval (like 5 min). When the host is turned off/on, the host is assumed to be consuming the idle power.

3.3 Discount Amount

As introduced in Sect. 1, the meaning of discount could be either the discount offered to users or the revenue loss of service providers that they charge lower price for services under brownout. In this paper, we note them as discount.

The total discount amount at time interval t is modeled as the sum of discount of all deactivated application components at t:

$$D(t) = \sum_{i=1}^{M} \sum_{j=1}^{N_i} d(VM_{i,j}(t)) \tag{4}$$

where $D(t)$ is the total discount amount at t that obtained from all VMs on hosts, N_i is the number of VMs assigned to server i, M is the number of servers. The individual discount $d(VM_{i,j}(t))$ is the sum of discount amount of deactivated application components $d(App_c)$ of $VM_{i,j}$, which is shown in Eq. (5):

$$d(VM_{i,j}) = \sum_{c=1}^{C_j} d(App_c) \tag{5}$$

where C_j is the number of application components hosted on VM_j, and only the deactivated components are counted.

3.4 Problem Definition

Let $Q(t) \in \mathbb{Q}$, where $\mathbb{Q} = \eta_1, \ldots, \eta_{|\mathbb{Q}|}$, $\eta_i \in \mathbb{Q}$. The $Q(t)$ is a combination of two vectors: energy consumption vector $E(t)$ and discount amount vector $D(t)$, representing the possible energy consumption and discount amount at different system states. Let $C(t)$ to be all the application component states at t, we have

Definition 1. *The system state at time interval t can be specified as:*

$$S(t) \triangleq [Q(t), C(t)] \tag{6}$$

The system state $S(t)$ contains the energy consumption and discount amount as well as their corresponding application components states.

At each time interval, we calculate the state information as:

$$g(t) = E(t) + \lambda D(t) \tag{7}$$

where λ is the weight of discount. The higher λ implicates that more weights are given to the discount amount. In the whole scheduling period T under policy π, our optimization objective is:

$$\min_{\pi} \qquad g(\pi) = \sum_{t=0}^{T} [E(t) + \lambda D(t)] \tag{8}$$

4 Proposed Approach

4.1 Approximate Markov Decision Process Model

To adopt the Markov model, we assume that the workload satisfies the Markov property, which means the state transitions caused by workloads are memoryless. Our experiments are conducted with Planetlab workload, which has been validated to satisfy Markov chain property [4]. In our model, we assume that the probability of application components to transfer their states at the next time period only depends on the workloads of the current time period and independent on earlier states. We formulate our problem as finite horizon MDP that we investigate a fixed length of time.

Then we can solve our objective function by using Bellman equation [3]:

$$V^*(S_i) = \arg\min_{\gamma \in \mathbb{R}} [g(S_i) + \sum_{S_j \in S} Pr[S_j | S_i, \gamma] V^*(S_j)] \tag{9}$$

$g(S_i)$ is the instant cost under system state S_i, and $V^*(S_i)$ is the expected energy consumption and discount obtained from S_j to S_i. We also denote $\gamma(t) \triangleq [\gamma_1(t), \ldots, \gamma_n(t)] \in \mathbb{R}$ as the operations (activation or deactivation actions) for application components. $V^*(S_i)$ can be found by iteratively obtaining minimum energy consumption and discount until convergence.

Let $\hat{p_{i,j}}$ denote the estimated transition probability that the application component changes its state. The transition probability is computed as:

$$\hat{p}_{i,j} = \sqrt{\frac{\hat{M}}{M}} \times Pr(\frac{u(App_c)}{d(App_c)} = z_C) \tag{10}$$

$Pr(\frac{u(App_c)}{d(App_c)} = z_C)$ is the probability that the ratio of component utilization and discount $\frac{u(App_c)}{d(App_c)}$ falls into category z_C. We divide the probability into C (the maximum number of components on a VM) categories. For all the components with the probability falls into the same category, we apply the same operation. To avoid the curse of dimension, noted by [11], we adopt key states to reduce state space. With key states, the component states on a VM is reduced to the maximum number of components on a VM as $|C|$. $\hat{M}$ is the estimated number of overloaded hosts, which is calculated based on a sliding window [5]. The advantage of sliding window is to give more weights to the values of recent time intervals. Let L_w to be the window size, and $N(t)$ to be the number of overloaded hosts at t, we estimate $\hat{M}$ as:

$$\hat{M}(L_w) = \frac{1}{L_w} \sum_{t=0}^{L_w - 1} N(t) \tag{11}$$

We denote the states as key states S_k as described above. With proof in [11], $\forall S_i \in S_k$ for all the VMs, the equivalent Bellman's equation in Eq. (9) can be approximately formulated as:

$$V^*(S_i) \approx \sum_{m=1}^{M} \sum_{n=1}^{N_m} (g(S_i) + \arg \min_{\gamma_n \in \mathbb{R}_n} \{ \sum_{S_j \in S_k} Pr[S_j | S_i, \gamma_n] \widetilde{V_n^*}(S_j) \}) \qquad (12)$$

The state spaces thus are reduced to polynomial with linear approximation. The M is the number of hosts and N_m is the number of VM assigned to server m.

4.2 Brownout Algorithm Based on Markov Decision Process (BMDP)

Our novel brownout algorithm is embedded within a VM placement and consolidation algorithm. We adopt the VM placement and consolidation algorithm (PCO) proposed in [4], which is also one of our baselines in Sect. 5.

The PCO algorithm is a heuristic to reduce energy consumption through VM consolidation. In the initial VM placement phase, PCO sorts all the VMs in decreasing order by their current CPU utilization and allocates each VM to the host that increases the least power consumption due to this allocation. In the VM consolidation phase, PCO optimizes VM placement by separately picking VMs from over-utilized and under-utilized hosts to migrate, and finding new placements for them. After migration, the over-utilized hosts are not overloaded any more and the under-utilized hosts are switched to sleep mode.

Our brownout algorithm based on approximate Markov Decision Process is shown in Algorithm 1 and includes 6 steps:

(1) System initialization (lines 1–2): Initializing the system configurations, including overloaded threshold TP, dimmer value θ_t, vector $\mathbb{Q}$ that contains the $D(t)$ and $E(t)$ information, as well as objective states $\mathbb{S}_d$, and applying VM placement algorithm in PCO to initialize VM placement.

(2) Estimating transition probability of each application component (lines 3–14): At each time interval, the algorithm firstly estimates the number of overloaded hosts. The dimmer value is computed as $\sqrt{\frac{\hat{M}}{M}}$, which is adaptive to the number of overloaded hosts. If no host is overloaded, the value is 0 and no component is deactivated. If there are overloaded hosts, the transition probabilities of application components are computed using Eq. (10).

(3) Finding the states that minimize the objective function (lines 15–17): Traversing all the key states by value iteration according to Eq. (12), where $D^{'}(t)$ and $E^{'}(t)$ are the temporary values at the current state.

(4) Updating system information (lines 18–20): The algorithm updates the obtained energy consumption and discount values if $g(t)$ in Eq. (7) is reduced, and records the optimized states. The current states are substituted by the state with lower $g(t)$.

(5) Deactivating the selected components (line 22): The brownout controller deactivates the selected components to achieve objective states.

(6) Optimize VMs placement (line 24): The algorithm uses the VM consolidation approach in PCO to optimize VM placement via VM consolidations.

Algorithm 1 Brownout based Markov Decision Process Algorithm (BMDP)

```
Input: host list $hl$ with size $M$, VM list, application components information, overloaded power threshold $TP$, dimmer value $\theta_t$ at time $t$, destination states $S_d(t)$, energy consumption $E(t)$ and discount amount $D(t)$ in $\mathbb{Q}$
Output: total energy consumption, discount amount
1: $TP \leftarrow 0.8$; $\theta_t \leftarrow 0$; $\forall E(t), \forall D(t) \in \mathbb{Q} \leftarrow max$; $S_d(t) \in \mathbb{S}_d \leftarrow NULL$
2: use PCO algorithm to initialize VMs placement
3: while true do
4:   for $t \leftarrow 0$ to $T$ do
5:     $\theta_t \leftarrow = \sqrt{\frac{\hat{M}_t}{M}}$
6:     for all $h_i$ in $hl$ do
7:       if $h_i$ is overloaded then
8:         for all $VM_{i,j}$ on $h_i$ do
9:           for all $App_c$ on $VM_{i,j}$ do
10:            $Pr(App_c) \leftarrow \theta_t \times Pr(\frac{u(App_c)}{d(App_c)} = z_C)$
11:          end for
12:        end for
13:      end if
14:    end for
15:    for all $S_j(t) \in S_k(t)$ do
16:      $V^*(S_i) = \sum_{m=1}^{m=M} \sum_{n=1}^{n=N_m} (g(S_i) + \min_{\gamma_n \in \mathbb{R}_n} \{\sum_{S_j \in S_k} Pr[S_j|S_i, \gamma_n] \widetilde{V_n^*}(S_j)\})$
17:      $g(t) = E^{'}(t) + \lambda D^{'}(t)$
18:      if $g(t) < E(t) + \lambda D(t)$ then
19:        $E(t) \leftarrow E^{'}(t)$ ; $D(t) \leftarrow D^{'}(t)$ ; $S_d(t) \leftarrow S_j(t)$
20:      end if
21:    end for
22:    deactivate the selected components to achieve state $S_d(t)$
23:  end for
24:  use VM consolidation in PCO algorithm to optimize VM placement
25: end while
```

The complexity of the BMDP algorithm at each time interval is consisted of the brownout part and VM consolidation part. The complexity of the transition probability computation is $O(C \cdot N \cdot M)$, where C is the maximum number of components in all applications, N is the maximum number of VMs on all the hosts and M is the number of hosts. With the key states, the space state of the MDP in brownout part is $O(C \cdot N \cdot M)$. According to Eq. (12), the actions are reduced to $O(C \cdot N \cdot M)$, so the overall MDP complexity is $O(C^2 \cdot N^2 \cdot M^2)$. The complexity of the PCO part is $O(2M)$ as analyzed in [4]. Therefore, the overall complexity is $O(C \cdot M \cdot N + C^2 \cdot N^2 \cdot M^2 + 2M)$ or equally $O(C^2 \cdot N^2 \cdot M^2)$.

5 Performance Evaluation

5.1 Methodology

We use the CloudSim framework [6] to simulate a cloud data center. The data center contains two types of hosts and four types of VMs that are modeled based on current offerings in EC2 as shown in Table 1. The power models of the adopted hosts are derived from IBM System x3550 M3 with CPU Intel Xeon X5670 and X5675 [1]. We set the time required to turn on/off hosts as 0.5 min.

We implemented application with optional components, and each component has its corresponding CPU utilization and discount amount. The components are uniformly distributed on VMs.

We adopt the realistic workload trace from more than 1000 PlanetLab VMs [17] to create an overloaded environment [5]. Our experiments are simulated under one-day scheduling period and repeated for 10 different days. The brownout is invoked every 5 min (one time interval) if hosts are overloaded. The sliding window size L_w in Eq. (11) to estimate the number of overloaded hosts is set as 12 windows (one hour).

The CPU resource is measured with capacity of running instructions. Assuming that the application workload occupies 85% resource on a VM and the VM has 1000 million instructions per second (MIPS) computation capacity, then it represents the application constantly requires $0.85 \times 1000 = 850$ MI per second in the 5 min time interval.

Table 1. Host/VM types and capacity

Name	CPU	Cores	Memory	Bandwidth	Storage
Host Type 1	1.86 GHz	2	4 GB	1 Gbit/s	100 GB
Host Type 2	2.66 GHz	2	4 GB	1 Gbit/s	100 GB
VM Type 1	2.5 GHz	1	870 MB	100 Mbit/s	1 GB
VMType 2	2.0 GHz	1	1740 MB	100 Mbit/s	1 GB
VM Type 3	1.0 GHz	1	1740 MB	100 Mbit/s	1 GB
VM Type 4	0.5 GHz	1	613 MB	100 Mbit/s	1 GB

We use three baseline algorithms for comparison as below:

(1) VM Placement and Consolidation algorithm (PCO) [4]: the algorithm has been described at the beginning of Sect. 4.2.

(2) Utilization-based Probabilistic VM consolidation algorithm (UBP) [7]**:** for VM initial placement, UBP adopts the same approach as PCO. For VM consolidation, UBP applies a probabilistic method [15] to select VMs from overloaded host. The probabilistic method calculates the migration probability $f_m(u)$ based on host utilization u as: $f_m(u) = (1 - \frac{u-1}{1-T_h})^{\alpha}$, where T_h is the upper threshold for detecting overloads and α is a constant to adjust probability.

(3) Brownout algorithm with Highest Utilization and Price Ratio First Component Selection Algorithm (HUPRFCS) [21]**:** it is a brownout-based heuristic algorithm. This algorithm deactivates the application components from the one with the highest $\frac{u(App_)}{d(App_c)}$ to the others with lower $\frac{u(App_c)}{d(App_c)}$ until the deactivated components obtain the expected utilization reduction, which is a deterministic algorithm. HUPRFCS is an efficient approach to reduce energy consumption under discount amount constraints.

To evaluate algorithms' performance, we mainly explore two parameters:

(1) Overloaded threshold: it identifies the CPU utilization threshold that determines the overloaded hosts, and it is varied from 80% to 95% in increments of 5%. We adopt this parameter since both [4] and [15] have shown that it influences energy consumption.

(2) Percentage of optional utilization in an application: it shows how much utilization in application is optional and can be deactivated. It is varied from 25% to 100% in increments of 25%. An application with 100% optional utilization represents that the application components or microservices are self-contained and each of them is allowed to be disabled temporarily (not disabling all the components at the same time), such as a stateless online document processing application. We assume the application maximum discount is identical to the percentage of optional utilization, for example, 50% optional utilization in an application comes along with 50% discount amount.

We assume that the optional components utilization $u(App_c)$ and discount $d(App_c)$ conform normal distribution $u(App_c){\sim}N(\mu, \sigma^2)$, $d(App_c){\sim}N(\mu, \sigma^2)$, the μ is the mean utilization of component utilization or discount, which is computed as the percentage of optional utilization (or discount amount) divided by the number of optional components. The σ^2 is the standard deviation of optional components utilization or discount. In our experiments, we consider both optional component utilization standard deviation and discount standard deviation are less than 0.1, which represents that the optional components are designed to have balanced utilization and discount.

5.2 Results

5.2.1 Comparison with Different λ

To investigate the impacts of different discount weights in Eq. (7), we conduct a series of experiments with different λ. In these evaluations, the hosts number and VMs number are set to 200 and 400 respectively, the overloaded threshold is set to 85% and the percentage of optional utilization is set to 50%. Figure 2 indicates that energy consumption increases and discount amount decreases when λ increases. The reason lies in that larger λ will guide our algorithm to find the states that offer less discount. From the results, we notice that when λ value is less than 4500, BMDP saves more energy than UBP and PCO, and in comparison to HUPRFCS, BMDP has similar energy consumption and reduces significant discount amount.

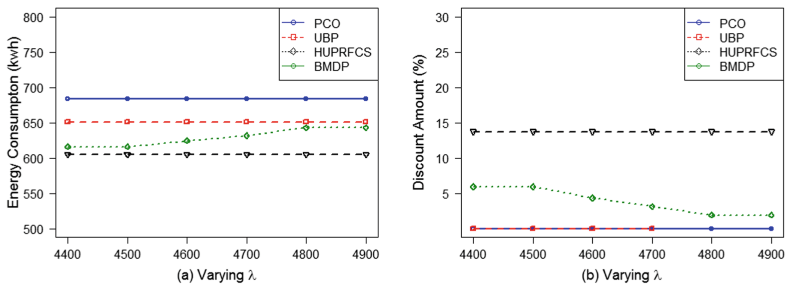

Fig. 2. Comparison with different λ. The parameter λ is the weight of discount.

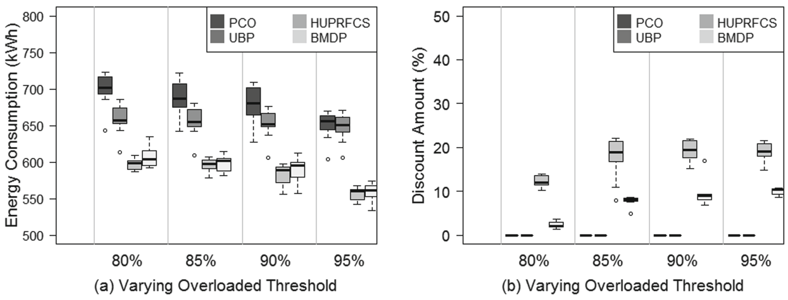

Fig. 3. Varying overloaded threshold

In the following evaluations, we set λ to a small value (i.e. $\lambda = 100$) so that the energy consumption of BMDP is below two baselines (PCO and UBP) and close to HUPRFCS. Additionally, with this λ value, the discount of BMDP is less than the discount produced by HUPRFCS.

5.2.2 Comparison Under Varied Overloaded Thresholds

The performance evaluated under different overloaded thresholds is shown in Fig. 3. Other parameters are configured as same as in Sect. 5.2.1. In Fig. 3(a), we observe that the energy consumption of all the algorithms are reduced when the overloaded threshold increases, for example, PCO-80% has 699.6 kWh with 95% Confidence Interval (CI) (682.6, 716.6) and reduces it to 649.9 kWh with 95% CI: (635.8, 664.1) in PCO-95%; BMDP-80% has 607.8 kWh with 95% CI: (598.1, 617.4) and saves it as 558.4 kWh with 95% CI: (549.6, 567.2) in BMDP-95%. The reason lies in that higher overloaded thresholds allow more VMs to be packed on the same host, so that more hosts are shutdown. When overloaded thresholds are between 80% to 90%, UBP reduces around 5% energy consumption compared to PCO, while HUPRFCS and BMDP save about 14–16% more energy consumption than PCO. When the overloaded threshold is 95%, PCO and UBP achieve close energy consumption, while HUPRFCS and BMDP still reduce around 16% energy compared with them.

As the energy consumption of HUPRFCS and BMDP are quite close, we conduct paired t-tests for HUPRFCS and BMDP as shown in Table 2. We notice that the differences between them are less than 2%, and when the overloaded thresholds are 85% and 95%, the *p-values* are 0.09 and 0.45 respectively, which indicates weak evidence to prove that they are different.

Comparing the discount amount, Fig. 3(b) shows that there is no discount offered in PCO and UBP, but HUPRFCS offers 11% to 20% discount and BMDP reduces it to 3% to 11% as the trade-off due to components deactivation. This is because, based on heuristics, HUPRFCS quickly finds the components with higher utilization and discount ratio, while BMDP steps further based on MDP to optimize the component selection.

Table 2. Paired T-Tests with 95% CIs for Comparing Energy Consumption by HUPRFCS and BMDP under Different Overloaded Thresholds

Algorithm 1 (kWh)	Algorithm 2 (kWh)	Difference (kWh)	***p-value***
HUPRFCS-80% (598.01)	BMDP-80% (607.78)	−9.77 (−15.14, −4.39)	0.0026
HUPRFCS-85% (595.87)	BMDP-85% (599.24)	3.37 (−0.77, 7.52)	0.099
HUPRFCS-90% (581.91)	BMDP-90% (587.97)	−6.05 (−9.41 −2.69)	0.0027
HUPRFCS-95% (557.03)	BMDP-95% (558.41)	−1.38 (−5.36, 2.6)	0.45

5.2.3 Comparison Under Varied Percentage of Optional Utilization

In Fig. 4, we compare the algorithms with different percentages of optional utilization. Other parameters are set the same as those in Sect. 5.2.1. As shown in Fig. 4(a), for PCO and UBP, their energy consumption are not influenced by different percentage of optional utilization. PCO has 684 kWh with 95% CI: (667.4, 700.6), and UBP has reduced 4.7% to 651.9 with 95% CI: (637.3, 666.5). Compared with PCO, HUPRFCS-25% reduces 11% energy to 605kWh with 95% CI: (596.6, 613.4), and BMDP-25% reduces 9% energy to 615.9 kWh with 95% CI: (605.9, 625.8). When the percentage of optional utilization increases, the more energy consumption is saved by HUPRFCS and BMDP. For instance, HUPRFCS-100% and BMDP-100% achieve around 20% energy saving as 556.9kWh with 95% CI: (550.9, 562.3) and 551.6kWh with 95% CI: (545.8, 557.4) respectively. The reason is that higher percentage of optional percentage allows more utilization to be reduced. For the discount amount comparison in Fig. 4(b), it shows that HUPRFCS offers 10% to 25% discount amount as trade-offs, while BMDP only offers 3% to 10% discount amount.

Because the energy consumption of HUPRFCS and BMDP are quite close, we conduct the paired t-test for HUPRFCS and BMDP as illustrated in Table 3. When the percentage of optional utilization are 75% and 100%, the *p-values* are 0.099 and 0.057, which indicates weak evidence to prove that they are different. And with other percentage of optional utilization, the energy consumption differences are less than 2%.

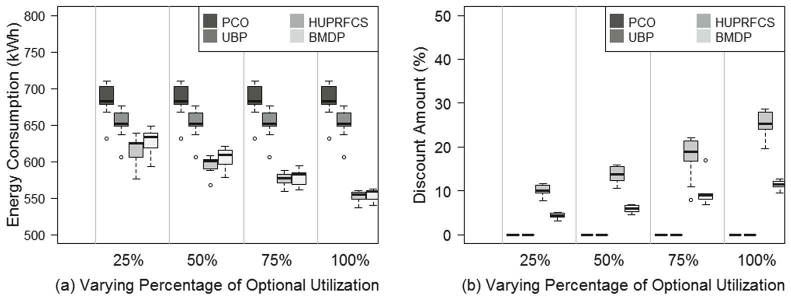

Fig. 4. Varying percentage of optional utilization

Table 3. Paired T-Tests with 95% CIs for Comparing Energy Consumption by HUPRFCS and BMDP under Different Percentage of Optional Utilization

Algorithm 1 (kWh)	Algorithm 2 (kWh)	Difference (kWh)	***p-value***
HUPRFCS-25% (617.57)	BMDP-25% (628.10)	−10.52 (−12.52, −8.52)	0.00082
HUPRFCS-50% (595.0)	BMDP-50% (605.88)	−10.88 (−15.26, −6.5)	0.00032
HUPRFCS-75% (575.87)	BMDP-75% (579.24)	−3.37 (−7.52 −0.78)	0.099
HUPRFCS-100% (551.56)	BMDP-100% (556.59)	−3.12 (−5.08, −1.16)	0.0057

6 Conclusions and Future Work

Brownout has been proven to be effective to solve the overloaded situation in cloud data centers. Additionally, brownout can also be applied to reduce energy consumption. In this paper, we introduced the brownout system model by deactivating optional components in applications or microservices temporarily. In the model, the brownout controller can deactivate the optional components or microservices to deal with overloads and reduce data center energy consumption while offering discount to users. We also propose an algorithm based on brownout and approximate Markov Decision Process namely BMDP, to find the components should be deactivated. The simulations based on real trace showed that BMDP reduces 20% energy consumption than non-brownout baselines and saves discount amount than brownout baseline. As future work, we plan to implement a brownout prototype based on Docker Swarm.

Acknowledgments. This work is supported by China Scholarship Council, Australia Research Council Future Fellowship and Discovery Project Grants. We thank Chenhao Qu, Adel Nadjaran Toosi and Satish Narayana Srirama for their valuable suggestions.

References

1. Standard performance evaluation corporation. http://www.spec.org/power-ssj2008/results/res2010q2/
2. Bawden, T.: Global warming: Data centres to consume three times as much energy in next decade, experts warn (2016). http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html
3. Bellman, R.: Dynamic programming and lagrange multipliers. Proc. Nat. Acad. Sci. **42**(10), 767–769 (1956)
4. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener. Comput. Syst. **28**(5), 755–768 (2012)
5. Beloglazov, A., Buyya, R.: Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. IEEE Trans. Parallel Distrib. Syst. **24**(7), 1366–1379 (2013)

6. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Experience **41**(1), 23–50 (2011)
7. Chen, Q., Chen, J., Zheng, B., Cui, J., Qian, Y.: Utilization-based VM consolidation scheme for power efficiency in cloud data centers. In: 2015 IEEE International Conference on Communication Workshop (ICCW), pp. 1928–1933. IEEE (2015)
8. Corradi, A., Fanelli, M., Foschini, L.: VM consolidation: a real case based on openstack cloud. Future Gener. Comput. Syst. **32**, 118–127 (2014)
9. Delforge, P.: Data center efficiency assessment - scaling up energy efficiency across the data center industry: evaluating key drivers and barriers (2014). https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf
10. Dürango, J., Dellkrantz, M., Maggio, M., et al.: Control-theoretical load-balancing for cloud applications with brownout. In: 53rd IEEE Conference on Decision and Control, pp. 5320–5327 (2014)
11. Han, Z., Tan, H., Chen, G., Wang, R., Chen, Y., Lau, F.C.M.: Dynamic virtual machine management via approximate markov decision process. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9, April 2016
12. Hanumaiah, V., Vrudhula, S.: Energy-efficient operation of multicore processors by DVFs, task migration, and active cooling. IEEE Trans. Comput. **63**(2), 349–360 (2014)
13. Kim, K.H., Beloglazov, A., Buyya, R.: Power-aware provisioning of virtual machines for real-time cloud services. Concurrency Comput. Pract. Experience **23**(13), 1491–1505 (2011)
14. Klein, C., Maggio, M., Årzén, K.E., Hernández-Rodriguez, F.: Brownout: building more robust cloud applications. In: Proceedings of the 36th International Conference on Software Engineering, pp. 700–711 (2014)
15. Mastroianni, C., Meo, M., Papuzzo, G.: Probabilistic consolidation of virtual machines in self-organizing cloud data centers. IEEE Trans. Cloud Comput. **1**(2), 215–228 (2013)
16. Newman, S.: Building Microservices. O'Reilly Media Inc., Sebastopol (2015)
17. Park, K., Pai, V.S.: CoMon: a mostly-scalable monitoring system for planetlab. ACM SIGOPS Operating Syst. Rev. **40**(1), 65–74 (2006)
18. Pecero, J.E., Huacuja, H.J.F., Bouvry, P., Pineda, A.A.S., Locés, M.C.L., Barbosa, J.J.G.: On the energy optimization for precedence constrained applications using local search algorithms. In: International Conference on High Performance Computing and Simulation (HPCS), pp. 133–139 (2012)
19. Tomás, L., Klein, C., Tordsson, J., Hernández-Rodríguez, F.: The straw that broke the camel's back: safe cloud overbooking with application brownout. In: International Conference on Cloud and Autonomic Computing, pp. 151–160 (2014)
20. Toosi, A.N., Vanmechelen, K., Ramamohanarao, K., Buyya, R.: Revenue maximization with optimal capacity control in infrastructure as a service cloud markets. IEEE Trans. Cloud Comput. **3**(3), 261–274 (2015)
21. Xu, M., Dastjerdi, A.V., Buyya, R.: Energy efficient scheduling of cloud application components with brownout. IEEE Trans. Sustain. Comput. **1**(2), 40–53 (2016)
22. Zheng, K., Wang, X., Li, L., Wang, X.: Joint power optimization of data center network and servers with correlation analysis. In: IEEE INFOCOM 2014-IEEE Conference on Computer Communications, pp. 2598–2606 (2014)