Energy-Aware Task Scheduling Using DVFS and On/Off Switching in Data Center

Haoran Ma[1], Qiang Liu[1], Hang Liu[1], Zaixing Sun[2], Chonglin Gu[1(✉)], and Hejiao Huang[1]

[1] Harbin Institute of Technology, Shenzhen 518071, China
{guchonglin,huanghejiao}@hit.edu.cn
[2] Peng Cheng Laboratory, Shenzhen 518055, China

Abstract. Traditionally, servers in high performance clusters are always keeping in active state to process the incoming tasks. However, it may cause energy wastage when there are few requests, or service rejections when there come bursts of requests exceeding the service capacity of the cluster. In this paper, we propose an online scheduling of the tasks and servers through dynamically scaling the computing capacity by leveraging DVFS and On/Off switching of servers in cloud environment to minimize the total energy consumption. The stochastic nature of task arrivals complicates decisions regarding when and how many servers to switch On/Off, while the energy consumption and delays associated with these transitions cannot be ignored. Although DVFS can instantaneously increase processing speed, it may also significantly increase energy consumption. To address these challenges, we first introduce a heuristic algorithm called ETA (Energy-aware Task Allocation, which allocates tasks to the servers with the least incremental energy consumption in priority of their urgency, while minimizing the number of active servers. We then propose a DQN-based algorithm to further optimize On/Off switching by intelligently incorporating predicted task numbers as environmental states using LSTM. Numerical experiments demonstrate that our approach significantly reduces energy consumption in cloud data centers.

Keywords: Energy-aware · DVFS · On/Off switching · Task Scheduling · Cloud Data Center

1 Introduction

To meet potential peaks in task requests, most data centers typically provision a surplus of servers, leading to considerable energy waste. Consequently, on-demand scaling of computing capacity through On/Off switching is essential for energy conservation in cloud computing. Additionally, DVFS (Dynamic Voltage Frequency Scaling) can be employed, as it allows for reducing processor voltage in proportion to its running speed if not violating service constraints [1]. Existing studies on energy saving in cloud environments have primarily focused on DVFS and server On/Off switching:

D.-S. Huang et al. (Eds.): ICIC 2025, LNCS 15845, pp. 517–528, 2025.
https://doi.org/10.1007/978-981-96-9872-1_43

- **On/Off switching.** Meisner et al. in [2] proposed Powernap, an energy-saving approach for server systems, but neglected the transition overhead between active and off states. Sun et al. in [3] proposed an online On/Off algorithm based on job interval expectations, which is optimal for exponential and uniform distributions but degrades under real-world traces. Khasyah et al. in [4] proposed a multi-agent Actor-Critic reinforcement learning algorithm for node switching decisions. Despite these efforts, relying solely on On/Off switching may lead to increased deadline violations, as servers cannot be awakened immediately.
- **DVFS.** Kim et al. [5] proposed an EDF-DVS method to schedule tasks with deadline constraints to save the energy of the processors. Gu et al. [6] studied dispatching requests to geo-distributed data centers using both DVFS and On/Off switching, but ignored the overhead caused by On/Off switching. In fact, servers can be scaled not only in number but also in processing speed, presenting an opportunity to reduce more energy consumption.

In this paper, we investigate the problem of energy-efficient task scheduling in cloud data centers through the combined use of DVFS and server On/Off switching. Our objective is to minimize the total energy consumption while meeting task deadlines. Specifically, we consider an online scenario where multiple user requests arrive at each scheduling period, each characterized by distinct arrival times, deadlines, and execution times. To achieve optimal energy savings, we determine: 1) which server each task should be allocated to; 2) when and at what frequency each task should start execution; 3) when each server should switch on or off. The challenges of our problem are twofold. First, the decisions regarding task allocation, DVFS and On/Off switching are interdependent and mutually restrictive. Second, the suddenness and uncertainty of the workload in the cloud environment complicate On/Off decisions, rendering classical reactive On/Off policies insufficient for achieving optimal performance.

To address the aforementioned challenges, we propose ETA for task allocation and a Deep Q-Network (DQN)-based [7] algorithm to dynamically control server power states. The main contributions of this paper are as follows:

- We design a heuristic algorithm for task allocation which jointly consider task deadline constraints with optimal energy-efficient frequency of each task. These tasks will finally be allocated to as few servers as possible, so that the idle ones will be switched off for energy saving.
- We propose a DQN-based reinforcement learning approach to dynamically adjust the number of active servers. The future workload predicted by LSTM [8] is incorporated into the DQN environment state, enabling the agent to effectively handle sudden and uncertain workloads.
- Simulations using real-world task traces are conducted to evaluate the performance of the proposed algorithm.

2 System Model

Suppose there is a data center with N homogeneous servers in total. In every scheduling period $\tau \epsilon \{1,2, \ldots, T\}$, there are M^{τ} independent tasks allocated in servers. Each task $i \epsilon \{1, 2, \ldots, M^{\tau}\}$, is associated with arrival time a_i, deadline d_i, execution time at the

highest running speed r_i. Let $x_{i,j}$ denote boolean variable indicating whether task i is allocated to server j. We assume that the frequency of a task is fixed during the execution process. Let $y_{i,k}$ denote boolean variable indicating whether task i is running at k-th power level, and K is the total number of frequency levels. Let e_i be the actual execution time of task i, and f_{max} is the maximum frequency of the server, and f_k is CPU frequency at level k, where $k \in \{1,2,\ldots,K\}$. Let b_i denote the actual begin time of task i. Let M_{fail} denote the number of tasks failed to be completed within the deadlines, R_{fail} denote the ratio of M_{fail} to the total number of tasks during the entire scheduling period, and R_{fail}^{max} denote the upper bound of R_{fail}. Let P^k denote the power of an active server at k th frequency level, $P_{on\to off}$, $P_{off\to on}$ and P_{off} denote the power of the server in shutdown transition, setup transition and off state. In scheduling period τ, the durations of server j at k th frequency level, in shutdown transition, in setup transition, and in off state are denoted as $\delta_{active}^{\tau,j,k}$, $\delta_{on\to off}^{\tau,j}$, $\delta_{off\to on}^{\tau,j}$ and $\delta_{off}^{\tau,j}$.

The power of a server in active state P_{active} includes dynamic power $P_{dynamic}$ mainly consumed by CPU, and base power P_{base} keeping normal running of a server, where $P_{active} = P_{dynamic} + P_{base}$. $P_{dynamic}$ is in proportion to its frequency and the square of voltage, which is approximately in linear relationship with frequency [5]. Thus, $P_{dynamic} = \alpha * f^3$, where α is the coefficient for dynamic power, and f is the CPU frequency of this server. Then, $P^k = \alpha * {f_k}^3 + P_{base}$.

In this paper, we aim to minimize the total energy consumption of servers by leveraging DVFS and On/Off switching of the servers to execute the tasks of user requests in an online scenario. Our problem can be formulated as follows:

$$\textbf{Minimize}: E = \sum_{\tau=1}^{T}\left(\sum_{j=1}^{N}\left[\begin{array}{l}\sum_{k=1}^{K} P^k * \delta_{active}^{\tau,j,k} + P_{on\to off} * \delta_{on\to off}^{\tau,j} \\ +P_{off\to on} * \delta_{off\to on}^{\tau,j} + P_{off} * \delta_{off}^{\tau,j}\end{array}\right]\right) \tag{1}$$

$$\textbf{Subjectto}: \sum_{j=1}^{N} x_{i,j} = 1 \tag{2}$$

$$\sum_{k=1}^{K} y_{i,k} = 1 \tag{3}$$

$$b_i + r_i * \frac{f_{max}}{\sum_{k=1}^{K}(y_{i,k} * f_k)} \le d_i \tag{4}$$

$$R_{fail} = M_{fail} / \sum_{\tau=1}^{T} M^{\tau} \le R_{fail}^{max} \tag{5}$$

In this formulation, each task can only be allocated to exactly one server, so we have constraint (2). Each task has a fixed execution frequency, so we have constraint (3). The finishing time of each task should be within its deadline, so we have constraint (4). We need to meet the QoS requirement, so we have (5).

3 Algorithm Design

3.1 Task Allocation

Tasks can be executed at **Optimal Frequency** with minimum energy consumption without considering the deadline constraint. Let E_i denote the required energy consumed by task i when executed at the frequency f until completion. It can be represented as follows:

$$E_i = \left(\alpha f^3 + P_{base}\right) r_i \frac{f_{max}}{f} \tag{6}$$

$$E_i'(f) = \frac{2\alpha f^3 - P_{base}}{f^2} f_{max} r_i \tag{7}$$

It can be observed that $f_{opt} = \sqrt[3]{\frac{P_{base}}{2\alpha}}$ is independent of the tasks. For ease of understanding task allocation, we have the following four definitions:

- **Latest Start Time (LST)**: It is defined as the latest permissible time at which a task must start execution to avoid violating its deadline constraint.
- **Server's Task Queue (or called queue)**: It refers to the sequence of tasks awaiting execution on a server, typically managed as a FIFO queue.
- **New Task Start Time (NTST)**: It refers to the earliest time at which a newly allocated task can start execution on a server. Let t_j^{NTST} denote the NTST of server j, it can be calculated as: 1) If there are already tasks in the server's task queue, t_j^{NTST} equals the completion time of the last task in the queue; 2) If the server's task queue is empty and the server is in active state, t_j^{NTST} equals the current time; 3) otherwise, t_j^{NTST} equals the earliest time server j switching to on state. Let $b_{i,j}$ denote the start time of task i if it is allocated to server j, then $b_{i,j} = t_j^{NTST}$.
- **Latest Optimal Start Time (LOST)**: It refers to the latest start time that the task can run at optimal frequency. Let t_i^{LOST} denote the latest optimal start time of task i, then $t_i^{LOST} = d_i - \frac{f_{max}}{f_{opt}} r_i$.

Let $f_{i,j}$ denote the execution frequency if task i is allocated to server j, it can be calculated as: If the task can start before or at t_i^{LOST}, $f_{i,j}$ equals f_{opt}; if it can start after t_i^{LOST} but before its LST, $f_{i,j}$ equals $f_{max}/\left(\frac{d_i - b_i}{r_i}\right)$; Otherwise, $f_{i,j}$ equals f_{max}.

Let $E_{i,j}$ denote the increased energy consumption if task i is executed by server j, $D_{off \to on}$ denote the delay of setup transition. When allocating tasks to servers in on state or off $\to$ on state, there is no need to set up a new server,$E_{i,j}$ equals $\left(\alpha f_{i,j}^3 + P_{base}\right)\frac{f_{max}}{f_{i,j}} r_i$; When allocating tasks to servers in off state or on $\to$ off state, another server needs to be set up, $E_{i,j}$ equals $\left(\alpha f_{i,j}^3 + P_{base}\right)\frac{f_{max}}{f_{i,j}} r_i + P_{off \to on} D_{off \to on}$.

As shown in Algorithm 1, we propose a heuristic algorithm called ETA. The input includes a set of unallocated tasks, S_{on}, $S_{\text{off}\to\text{on}}$, S_{off} and $S_{\text{on}\to\text{off}}$ in scheduling period τ.

Algorithm 1: Energy-aware Task Allocation (ETA)

Input: $TSet$: set of unallocated tasks;
S_{on}: set of servers in active state or off→on state;
$S_{off\to on}$: set of servers in off→on state;
S_{off}: set of servers in off state;
$S_{on\to off}$: set of servers in on→off state;

Output: $x_{i,j}$: the 0-1 indicator showing whether task i is allocated onto server j;
b_i: actual start time of task i;
f_i^{actual}: actual execution frequency of task i;
$t_{off\to on}^{\tau,j}$: the time to setup server j in scheduling period τ;

1 Sort $TSet$ in ascending order of latest start time
2 Sort $S_{on}, S_{off\to on}, S_{off}, S_{on\to off}$ in ascending order of NTST
3 **for** $i \in TSet$ **do**
4 Calculate t_i^{LOST}
5 $OptServers \leftarrow$ Find the server set in $S_{on} \cup S_{off\to on}$ where task i can be executed at optimal frequency
6 **if** $OptServers \neq \varnothing$ **then**
7 $j_0 \leftarrow$ Find the server with the latest NTST in $OptServers$
8 $x_{i,j_0} \leftarrow 1$
9 Calculate b_i and f_i^{actual}
10 **else**
11 $j_1 \leftarrow$ Find the server with the earliest NTST in $S_{on} \cup S_{off\to on}$
12 $j_2 \leftarrow$ Find the server with the earliest NTST in $S_{off} \cup S_{on\to off}$
13 Calculate $E_{j_1}$and E_{j_2}
14 **if** $E_{j_1} < E_{j_2}$ **then**
15 $x_{i,j_1} \leftarrow 1$
16 Calculate b_i and f_i^{actual}
17 **else**
18 $x_{i,j_2} \leftarrow 1$
19 Calculate b_i and f_i^{actual}
20 **if** $j_2 \in S_{off}$ **then**
21 $S_{off} = S_{off} \setminus \{j_2\}$
22 $S_{on\to off} = S_{on\to off} \cup \{j_2\}$
$t_{off\to on}^{\tau,j_2} \leftarrow$ the current decision moment
23 **else**
24 $t_{off\to on}^{\tau,j_2} \leftarrow$ the moment it is just in off state

25 **return** $x_{i,j}, b_i, f_i^{actual}, t_{off\to on}^{\tau,j}$

The output includes: the allocation decision $x_{i,j}$, actual begin time b_i, actual execution frequency f_i^{actual} of each task i, and $t_{off\to on}^{\tau,j}$ (the time to set up server j in period τ). In this algorithm, we first determine the priority of tasks by their LST, ensuring that more urgent task has higher priority (line1). Then, we respectively sort the servers in

S_{on}, $S_{\text{off}\to\text{on}}$, S_{off} and $S_{\text{off}\to\text{on}}$ based on their NTST (line2). Let *OptServers* denote the set of such servers in $S_{\text{on}} \cup S_{\text{off}->\text{on}}$ that can execute task i at the optimal frequency.

If *OptServers* is not empty (line6), in order to allocate as many tasks as possible to the minimum number of servers, we allocate task i to the server in *OptServers* with the largest NTST (line7–8). Then b_i and f_i^{actual} are calculated (line9).

If *OptServers* is empty (line10), find a server j_1 and a server j_2 with the smallest NTST in $S_{\text{on}} \bigcup S_{\text{off}\to\text{on}}$ and $S_{\text{off}} \bigcup S_{\text{on}\to\text{off}}$, respectively (line11–12). Let E_{j_1} and E_{j_2} denote the energy consumed by server j_1 and j_2, respectively. If $E_{j_1} < E_{j_2}$, the task will be allocated to server j_1 (line14–16). Otherwise, the task will be allocated to server j_2 (line17–19). Given that server j_2 is in off state or off $\to$ on state, it needs to be set up considering the following two cases: If server j_2 is in the off state, it will be set up immediately, and change its state to off $\to$ on state. $t_{off\to on}^{\tau,j}$ is equal to the current decision moment (the beginning time of the current scheduling period) (line20–22); If server j_2 is in off $\to$ on state, no state change is needed, as it will remain in on $\to$ off state during decision until completely transited to off state. $t_{off\to on}^{\tau,j}$ is equal to the moment it just completes the transition to the off state (line23–24).

3.2 DQN-Based On/Off Scheduling Algorithm

In each scheduling period, the central controller as agent observes the resource states of the cloud computing center and then makes On/Off decisions. The whole scheduling process can be modeled as an MDP. Let S_τ denote the observations of agent in scheduling periodτ, it can be represented as:$S_\tau = \left[\lambda_1, \ldots, \lambda_i, \ldots \lambda_L, s_1, \ldots, s_j, \ldots s_N\right]$, where λ_i denotes the predicted task number of next i th ($1 \le i \le L$) scheduling periods. We adopt LSTM to make prediction. Let s_j represents the states of server$j$, it can be denoted as: $s_j = \left[state_j, span_j, t_j^{NTST}\right]$,where $state_j$ is the initial power state of server j inτ; $span_j$ is the span that server j has already been in$state_j$. The action set is designed as:$A = \{0, 1, \ldots, N, \ldots, 2N\}$. Let a_τ denote the On/Off action to be taken by our agent in periodτ,if $a_\tau = 0$, no On/Off operations will be performed;if $1 \le a_\tau \le N$, set up a_τ servers;if $N + 1 \le a_\tau \le 2N$, shut down $a_\tau - N$ servers; Considering the goal of minimizing the total energy consumption, the reward r_τ in period τ is designed as:$r_\tau = \varphi E_\tau + \psi M_{fail}^\tau$, where φ and ψ are constant coefficients, E_τ denotes the total energy consumption during periodτ, and M_{fail}^τ denotes the number of tasks that fail to meet their deadlines within periodτ.

We employ an Action Mask [9] to filter out invalid action in two stages:1) Action select stage. Before the agent takes an action, the Action Mask will generate a set of valid actions A_{legal}. 2) Q network update stage. When updating Q network, since invalid actions corresponding to $S_i{}'$ remain close to their initial values that are near 0 and the Q-values for valid actions will converge to a large negative value during training. If the Q-values of invalid actions in target network are not masked during updates, the target network may incorrectly use these invalid Q-values as estimates for $\max_{a'} Q_\omega\left(S_i{}', a'\right)$, making it difficult for the training to converge.

The details of DQN training are given in Algorithm 2. First, the replay buffer and environment are initialized (line1–3). At the beginning of each scheduling period τ, the

agent observes the environment and obtains the current state S_τ (line6). The Action-Mask module generates a set of valid actions A_{legal} based on current state (line 7). Based on ε-greedy, when the randomly generated value δ is larger than ε, the agent takes an action in A_{legal} with maximum Q value; otherwise, take an action in A_{legal} at random (line8–11). Subsequently, the environment will update its state, and feedback a reward to the agent (line12). The tuple (S, a, S', r) is then stored in the replay buffer (line13). The Q-network is trained using gradient descent, so that the parameter θ of Q network can be updated (line14–15). The parameter θ' of target network is updated by θ every *updateNum* steps (periods) (line16–17). This process continues until network training is complete.

Algorithm 2: DQN Training

Input: *Agent*: DQN;
env: virtual environment
Output: θ: the parameter of Q network;
θ': the parameter of target network;

1 Initialize replay buffer
2 **for** *episode* $\leftarrow 1$ *to* M **do**
3 Initialize environment *env*
4 $\tau \leftarrow 1$;
5 **while** $\tau \leq T$ **do**
6 $S_\tau \leftarrow$ Get the state of environment *env*
7 $A_{legal} \leftarrow ActionMask(S_\tau)$
8 **if** $\delta > \epsilon$ **then**
9 $a_\tau \leftarrow \arg\max_{a \in A_{legal}} Q(S_\tau, a)$
10 **else**
11 $a_\tau \leftarrow random(A_{legal})$
12 $r_\tau, S'_\tau \leftarrow$ Update environment *env*
13 Store $(S_\tau,\ a_\tau,\ S'_\tau,\ r_\tau)$ to replay buffer
14 **if** *buffer.size* $>$ *batchsize* **then**
15 $\theta \leftarrow$ Update Q Network
16 **if** $\tau \% updateNum == 0$ **then**
17 $\theta' \leftarrow \theta$
18 $\tau \leftarrow \tau + 1$

4 Experiment

We use the traces of tasks from Parallel Workloads Archive for simulation [10]. The entire experiment set is divided into training, validation, and test sets in a 0.6:0.2:0.2 ratio. In this experiment, four 100,000-s request traces are selected from the test set of each of the four datasets in the dataset collection, namely DataSet1, DataSet2, DataSet3, and DataSet4. The execution time at max frequency of the tasks can be get in [10]. The deadlines of the tasks are randomly selected from 0 to 5 times more than the execution time of each task at peak frequency.

In our experiment, the configuration parameters of the power management are from AMD Athlon server [11], which has 5 frequency levels, as can be seen in Table 1. Based

on literature [12], we assume the delay and power during setup and shutdown transitions are 20s, 10s, 110W, 90W. The power consumption of the server in idle consumes 53W. In our scheduling, the size of each scheduling period is 10 s and the number of servers is settled as 100.

Table 1. The parameters of servers

Frequency (GHz)	Dynamic Power (W)	Total Power (W)
0.8	2.1	55.1
1.5	15.1	68.1
2.0	35.9	88.9
2.7	88.3	141.3
3.0	121.1	174.1

During training, the exploration rate ε linearly decays from 1 to 0.05 over the first 50,000 time steps, and remains constant thereafter; the size of replay buffer is set as 50,000; the discount factor is set as 0.99; reward coefficient φ and ψ are set as -1 and -5000, respectively; the batch size is set as 64; the learning rate is set as 0.0001; the update interval of target network *updateNum* is set as 200 steps.

The most commonly used task allocation methods are as follows: FF (First Fit): It always allocates the task to the first server which can satisfy task deadline constraint in order of task arriving; BFD (Best Fit Decreasing) [13]: It always allocates the task with the smallest LST to the server with the largest NTST. If none of the active servers can meet the deadline constraint, a new server will be set up, and the task is allocated to it.

The most commonly used On/Off methods are as follows: Reactive will shut down the server if it is idle and it will set up the server when no active server can satisfy the task deadline constraint; SoftReactive will shut down the server if the server has been idle for fixed time which is set to be $T_{setup} * \frac{P_{peak}}{P_{idle}}$ in literature [5]; Probability-Based [3] will calculate the expectation of job interval for each server according to the historical data. The server will keep on if the energy consumption of being idle for the expectation of interval is smaller than the energy of shutting it down and setting it up. Multiagent-Based [4] will set an agent on each which independently decide whether to turn the server on or off. The environment variables are set in accordance with the original literature.

ARIMA, Dlinear and Exponential Smoothing are selected for comparison. Table 2 presents the performance of each prediction model across the four datasets, using Mean Squared Error (MSE) and Mean Absolute Error (MAE) to evaluate the effectiveness of the prediction algorithms in forecasting task loads. The results indicate that LSTM achieves the highest average accuracy across the four datasets.

To further evaluate the impact of prediction accuracy on scheduling, we conducted simulations using four different prediction models and compare their performance in terms of total energy consumption and task timeout ratio, as illustrated in Fig. 1. LSTM performs the best among them. We established two control groups: one using actual task numbers as prediction inputs (known-as-predict group) and the other using random

numbers as prediction inputs (random group). The results show that the LSTM-based approach outperforms the random group but does not match the performance of the known-as-predict group. This result validates the effectiveness of incorporating predicted task numbers into the DQN state and highlights the significant impact of prediction accuracy on the overall performance of the DQN-based scheduling system.

Table 2. Comparison of Prediction Accuracy for LSTM, DLinear, Informer and Arima

	LSTM		ARIMA		Dlinear		ES	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
DataSet1	**0.0328**	**0.0911**	0.0915	0.1278	0.0307	0.0938	0.0516	0.1017
DataSet2	**0.0281**	**0.0846**	0.0791	0.1137	0.0339	0.0922	0.0329	0.0942
DataSet3	**0.0174**	**0.0875**	0.0638	0.1035	0.0286	0.0764	0.0377	0.0927
DataSet4	**0.0412**	**0.0857**	0.0724	0.1158	0.0351	0.0935	0.0601	0.0987

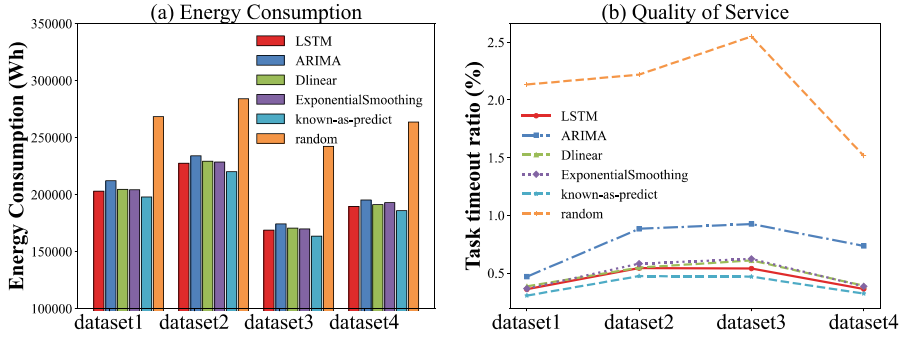

Fig. 1. Comparisons of different prediction algorithm performance.

Then we evaluate and select the most effective task allocation algorithm for each On/Off scheduling method. Figure 2, Fig. 3, Fig. 4, Fig. 5 illustrate the performance of the On/Off algorithms under various task allocation strategies, focusing on energy consumption and task timeout ratio. ETA is chosen for most On/Off methods because it effectively balances both server state transition energy and task execution energy. However, when combined with the Reactive algorithm, which immediately shuts down idle servers, ETA's strategy of allocating tasks to the fewest servers can lead to frequent shutdowns and subsequent task timeouts. This is because ETA often produces idle servers, which Reactive immediately shuts down, causing subsequent tasks to timeout and increasing the frequency of server state transitions. BFD, by allocating tasks to as many servers as possible, minimizes the number of idle servers, thereby reducing the likelihood of Reactive performing shutdown operations. Therefore, we select the task allocation algorithm for each On/Off method as follows: Reactive + BFD, SoftReactive + ETA, Probability-Based + ETA and Multiagent-Based + ETA.

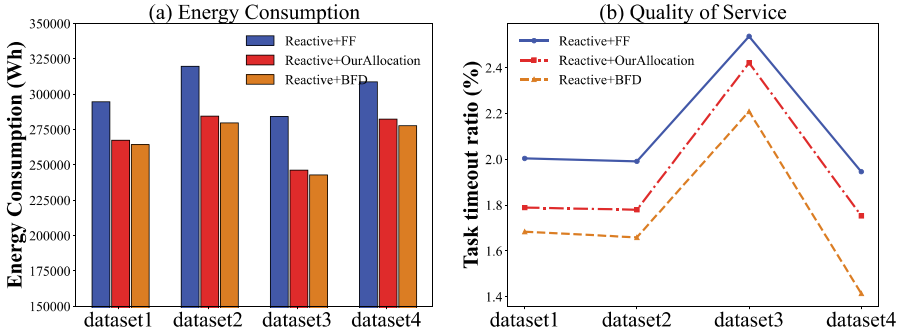

Fig. 2. Comparisons of different task allocation algorithm for Reactive.

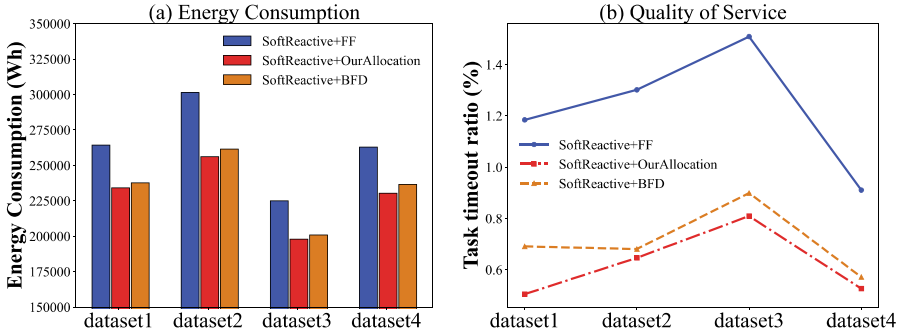

Fig. 3. Comparisons of different task allocation algorithm for SoftReactive.

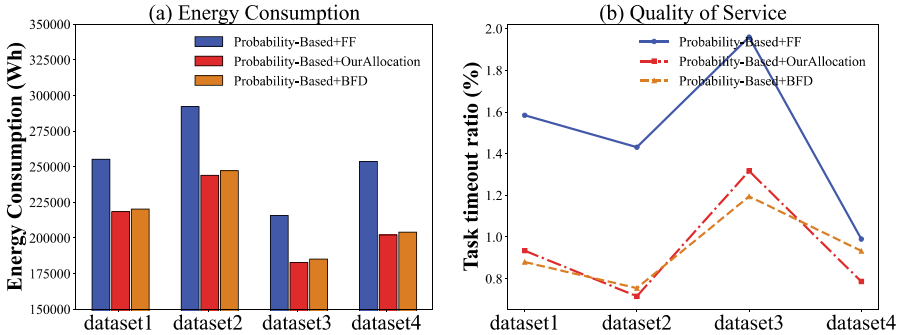

Fig. 4. Comparisons of different task allocation algorithm for Probability-Based.

Based on the selected task allocation algorithm, we compare the On/Off methods in terms of energy consumption and task timeout ratio. The experimental results, as shown in Fig. 6, indicate that DQN algorithm achieves the lowest energy consumption and the lowest task timeout ratio. The Reactive performs the worst in terms of both energy consumption and QoS. The SoftReactive has a low task reject ratio at the expense of

increased energy consumption. However, it fails to preemptively set up some servers in anticipation of future peak tasks. As a result, its task timeout ratio remains higher than that of our method. The Probability-Based performs poorly in real trajectories with high task uncertainty. The Multiagent-Based algorithm faces challenges in resolving conflicts between individual agent goals and overall agent objectives. Consequently, individual agent trained under this algorithm tends to be off state and reject tasks, leading to high task timeout ratio. Our method achieves an average energy savings of 15.6% compared to other works while with higher service quality.

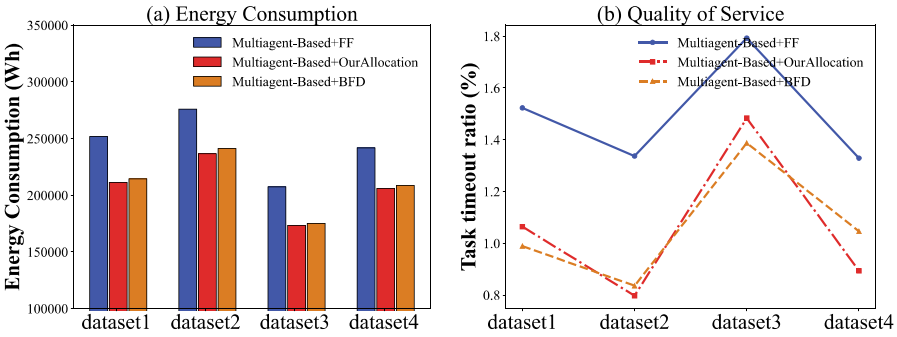

Fig. 5. Comparisons of different task allocation algorithm for Multiagent-Based.

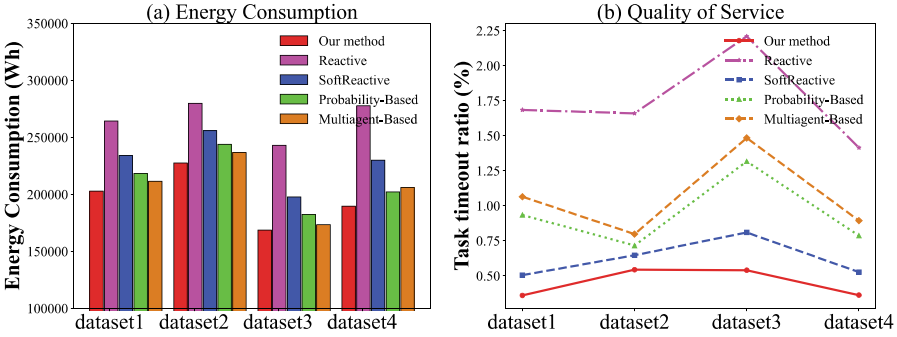

Fig. 6. Comparisons of different On/Off algorithm performance.

5 Conclusion

In this paper, we have investigated the problem of energy-efficient task scheduling in cloud data centers by leveraging DVFS and server On/Off switching. Specifically, we proposed a heuristic algorithm, ETA, which allocates tasks based on their urgency and incremental energy consumption. This approach effectively minimizes the number of active servers and overall energy consumption. Furthermore, we developed a DQN-based reinforcement learning method to dynamically adjust the number of active

servers, thereby further reducing total energy consumption. Notably, we incorporated LSTM-predicted workload into the environment states of our DQN model to enhance the accuracy and adaptability of our scheduling strategy. Extensive simulations using real-world task traces demonstrate that our proposed methods significantly outperform existing approaches in terms of energy efficiency and QoS.

Acknowledgments. This work is financially supported by Shenzhen Science and Technology Program under Grant No. GXWD20220817124827001 and No. JCYJ20210324132406016.

References

1. Miftakhutdinov, R., Ebrahimi, E., Patt, Y.N.: Predicting performance impact of DVFS for realistic memory systems. IEEE/ACM International Symposium on Microarchitecture, pp. 155–165 (2012)
2. Meisner, D., Gold, B.T., Wenisch, T.F.: PowerNap: eliminating server idle power. ACM Sigplan Notices **44**(3), 205–216 (2009)
3. Sun, J., Yang, Q., Yang, Z.: Probability-based online algorithm for switch operation of energy efficient data center. IEEE Transactions on Cloud Computing **10**(1), 608–618 (2019)
4. Khasyah, F.R., Santiyuda, K.G., Kaunang, G., et al.: An advantage actor-critic deep reinforcement learning method for power management in HPC systems. International Conference on Parallel and Distributed Computing: Applications and Technologies, pp. 94–107 (2022)
5. Kim, K.H., Buyya, R., Kim, J.: Power aware scheduling of bag-of-tasks applications with deadline constraints on DVFS-enabled clusters. International Symposium on Cluster Computing and the Grid, pp. 541–548 (2007)
6. Gu, L., Zeng, D., Barnawi, A., et al.: Optimal task placement with QoS constraints in geo-distributed data centers using DVFS. IEEE Trans. Comput. **64**(7), 2049–2059 (2015)
7. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Playing Atari with Deep Reinforcement Learning. arXiv preprint arXiv:1312.5602 (2013)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
9. Huang, S., Ontañón, S.: A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. arXiv preprint arXiv:2006.14171 (2020)
10. Feitelson, D.G., Tsafrir, D., Krakov, D.: Experience with using the parallel workloads archive. J. Parallel and Distributed Computing **74**(10), 2967–2982 (2014)
11. Li, J., Li, Z., Ren, K., et al.: Towards optimal electric demand management for internet data centers. IEEE Transactions on Smart Grid **3**(1), 183–192 (2012)
12. Duy, T.V.T., Sato, Y., Inoguchi, Y.: Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. IEEE International Symposium on Parallel & Distributed Processing, pp. 1–8 (2010)
13. Azizi, S., Zandsalimi, M., Li, D.: An energy-efficient algorithm for virtual machine placement optimization in cloud data centers. Clust. Comput.Comput. **23**(4), 3421–3434 (2020)