This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

A Double Deep Q-learning Model for Energy-efficient Edge Scheduling

Qingchen Zhang, Man Lin, Laurence T. Yang, Zhikui Chen, Samee U. Khan, and Peng Li

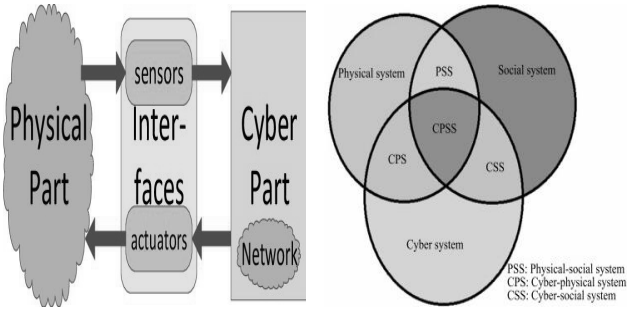
Abstract—Reducing energy consumption is a vital and challenging problem for the edge computing devices since they are always energy-limited. To tackle this problem, a deep Q-learning model with multiple DVFS (dynamic voltage and frequency scaling) algorithms was proposed for energy-efficient scheduling (DQL-EES). However, DQL-EES is highly unstable when using a single stacked auto-encoder to approximate the Q-function. Additionally, it cannot distinguish the continuous system states well since it depends on a Q-table to generate the target values for training parameters. In this paper, a double deep Q-learning model is proposed for energy-efficient edge scheduling (DDQ-EES). Specially, the proposed double deep Q-learning model includes a generated network for producing the Q-value for each DVFS algorithm and a target network for producing the target Q-values to train the parameters. Furthermore, the rectified linear units (ReLU) function is used as the activation function in the double deep Q-learning model, instead of the Sigmoid function in QDL-EES, to avoid gradient vanishing. Finally, a learning algorithm based on experience replay is developed to train the parameters of the proposed model. The proposed model is compared with DQL-EES on EdgeCloudSim in terms of energy saving and training time. Results indicate that our proposed model can save average $2\% - 2.4\%$ energy and achieve a higher training efficiency than QQL-EES, proving its potential for energy-efficient edge scheduling.

Index Terms—Edge computing; Energy saving; Deep Q-learning; Dynamic voltage and frequency scaling; Rectified linear units.

1 Introduction

Recent years have seen the continuous growth of Internet of Things and cyber-physical-social systems as a result of the great development of network techniques, especially wireless sensor networks [1], [2], [3]. Internet of Things integrate many physical and sensing devices including **RFID (Radio Frequency Identification)**, camera and so on to the cyber space while cyber-physical-social systems combine the human behaviors and social relations to the Internet of Things [4], [5]. Therefore, cyber-physical-social systems can be viewed as the generalization of Internet of Things that are called cyber-physical systems in this case. Fig. 1 shows the relation of cyber-physical systems and cyber-physical-social systems.

In Internet of Things and cyber-physical-social systems, a large quantity of information, also called big data, is collected by physical and sensing devices, and then the collected data is transmitted to data centers for processing and analytics [6], [7], [8]. **Typical big data collected from Internet of Things and cyber-physical-social systems includes state parameters such as temperature and pressure that can reflect the work state of the physical devices in the industrial manufacturing and traffic data that can be used for traffic prediction in the intelligent transportation.** Internet of Things and cyber-physical-social systems have achieved widely successful applications in industrial manufacturing and intelligent transportation with the big data analytics [9], [10]. **The widely used data analysis technologies include clustering that can be used to detect the abnormal work states of the physical devices and machine learning that can be used for moving object recognition and tracking in the intelligent traffic monitoring system.** Most of typical data centers have been employing cloud computing to process



(a) Cyber-physical systems (b) Cyber-physical-social systems

Fig. 1. Cyber-physical systems and cyber-physical-social systems.

- *Q. Zhang and L. T. Yang are with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, 611731, China, and the Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada. E-mail: ltyang@gmail.com*
- *M. Lin is with the Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada. E-mail: ltyang@gmail.com*
- *Z. Chen and P. Li are with the School of Software Technology, Dalian University of Technology, Dalian, China.*
- *S. U. Khan with the Department of Electrical and Computer Engineering at the North Dakota State University, Fargo, ND, USA.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

and analyze big data **since cloud computing can provide the powerful computing resources and the huge storage space** [11], [12], [13]. **However, with the increasing volume of collected data from Internet of Things and cyber-physical-social systems, cloud computing cannot satisfy the requirement of big data processing and analytics any longer due to the following reasons. First, a large volume of data is continually collected from the end of the network, posing a challenge on the network bandwidth. Specially, the data transmission speed has become a bottleneck for the cloud computing to process a large quantity of collected data in real time since the bandwidth between the network end and the cloud data centers is not large enough. Next, cloud computing cannot ensure the privacy protection sometimes in Internet of Things and cyber-physical-social systems [14]. For example, some medical information collected from wearable health systems is private. Transmitting the medical information to cloud for processing may leak the personal privacy, possibly causing a serious threat on personal life and property. Finally, most of sensing devices for collecting data in Internet of Things and cyber-physical-social systems are too limited in energy to upload the massive collected data to cloud. To overcome the above drawbacks of cloud computing, moving the data computation from the cloud to the edge is becoming a significant trend [14].**

Edge computing can be defined as any enabling technique that allows computation and storage to be executed at the network end. Therefore, the tasks should be moved to the proximity of information sources, also called things side, in the edge computing. "Edge" refers to the computing and storing resources between information sources and cloud. In particular, a smart phone, a gateway and a cloudlet can be viewed as edge resources when they are involved in data computing. In edge computing, data consumers not only require **services** from the data centers but also generate information while the data is generated by only data producers in the cloud computing paradigm.

Typically, most of edge computing resources such as smart phones, tablet computers and gateways are usually limited in energy. Specially, the limited energy supplying by the battery restricts the usage time and the service quality of edge computing devices for big data processing [15]. For example, the tasks will be forced to terminate once the energy of edge computing devices is used up. This case will trigger a serious disaster in security-sensitive areas such as aeronautical systems and medical treatment when the large-scale tasks are performed on edge computing devices. Therefore, it has become a crucial and challenging topic to extend the utility time for edge computing devices to provide the reliable services. One obvious way is to increase the battery capacity while the other way is to decrease the energy consumption. However, it is hard to increase the battery capacity because of the physic-chemical property of the battery. So, how to decrease the energy consumption has aroused great attention recently.

The energy of an edge computing device is mainly consumed by its processor(s). For instance, the processors of a Dell labtop consume more than half of its energy, as presented in Fig. 2 [16].

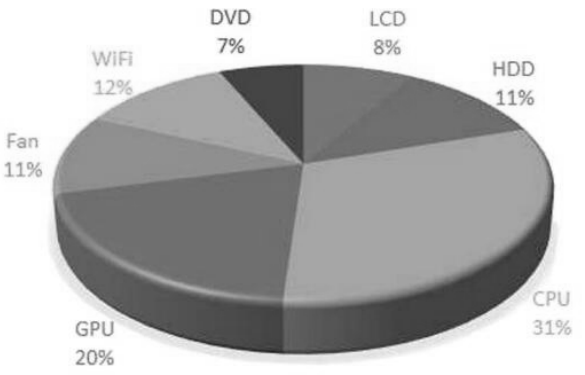


Fig. 2. Energy consumption of one Dell labtop.

In this work, we aim to reduce the energy consumption of the processor(s) for extending the utility time of edge computing devices. The energy consumption of the processor(s) includes two parts, i.e., static energy consumption and dynamic energy consumption. **The static energy is typically consumed by the leakage current while the dynamic energy is usually consumed by switching activities.** Specially, this paper focuses on the reduction of the dynamic energy consumption for edge computing devices since it often dominates the total energy consumption. Furthermore, we concentrate on saving the dynamic energy for the periodically real-time tasks by presenting an energy-efficient edge scheduling algorithm since many tasks run in edge computing devices are periodic and real-time.

To reduce the energy consumption, a deep Q-learning model was presented to schedule the periodic tasks in real-time systems (DQL-EES) by integrating three different DVFS algorithms, i.e., **cycle-conserving (CC), look-ahead (LA) and dynamic reclaiming algorithm (DRA)** [15]. In this scheme, the deep Q-learning model is implemented by integrating a deep learning algorithm with a Q-learning model to estimate the Q-value for every DVFS algorithm according to the current system state that is characterized by the system utilization and the dynamic slack. At the end of each hyperperiod, the DVFS algorithm with the lowest Q-value will be chosen to modify voltage and frequency for the next hyperperiod. Simulation results indicate that DQL-EES could reduce more energy consumption than any single DVFS algorithm. In DQL-EES, the Q-values produced by the Q-learning model are used as the target values to update the parameters of the deep neural network. However, DQL-EES is highly unstable when using a single stacked auto-encoder to approximate the Q-function. Additionally, it cannot distinguish the continuous system states well since it depends on a Q-table for providing the target values. **For instance, it views two different system sates** $\{SU = 0.512, DS = 0.624\}$ **and** $\{SU = 0.517, DS = 0.641\}$ **as the same system state with** $\{SU = 0.5, DS = 0.6\}$ **with the step size of 0.1. Although narrowing the step**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

size can improve the distinguishing ability, it will make the Q-table significantly larger and thus reduces the learning efficiency.

Aiming at this problem, a double deep Q-learning model is proposed for energy-efficient edge scheduling (DDQ-EES). Particularly, the double deep Q-learning model includes a generated network for producing the Q-value for every DVFS algorithm and a target network for producing the target Q-values to train the parameters. Furthermore, the rectified linear units (ReLU) function is used as the activation function in the double deep Q-learning model to avoid gradient vanishing [17]. Finally, a learning algorithm based on experience replay is developed to train the presented model. The proposed model is compared with DQL-EES in terms of energy saving and training time on EdgeCloudSim that is an environment for performance evaluation of edge computing applications. Results indicate that the presented model is able to save average $2\%-2.4\%$ energy and achieve a higher training efficiency than QQL-EES, proving its potential for energy-efficient edge scheduling.

The paper presents the following three contributions in edge computing:

- A double deep Q-learning model with a generated network and a target network is proposed to schedule the periodically real-time tasks of edge computing devices. The former aims to produce the Q-value for each DVFS algorithm while the latter is employed to produce the target Q-values for training the parameters of the presented double deep Q-learning model.
- Since the rectified linear units (ReLU) function is more efficient for gradient propagation than the Sigmoid function and ReLU can avoid gradient vanishing effectively, so we use ReLU as the activation function in the double deep Q-learning model, instead of the Sigmoid function, to produce more objective Q-value for each DVFS algorithm.
- A learning algorithm based on experience replay is developed to update the parameters of the presented model. Specially, the experience replay provides the system state transitions as the training samples.

The reminder of this paper is organized as follows. Section 2 reviews the related works and Section 3 presents the system models including the task model and the energy model used in this study. The presented double deep Q-learning model for energy-efficient edge computing is described in Section 4 and the learning algorithm for training the presented model is illustrated in Section 5. Section 6 reports the simulation and experimental results and Section 7 concludes the paper and discusses the future work.

2 Related Works

This work aims to decrease the dynamic energy consumption for the real-time systems with periodically real-time tasks in edge computing devices. Therefore, this section reviews the existing methods based on dynamic voltage and frequency scaling (DVFS) that is the most commonly employed technology for saving the dynamic energy. Specially, the representative DVFS algorithms are first described, and then the combined DVFS algorithms are reviewed.

2.1 Representative DVFS Algorithm

Some DVFS algorithms were presented to decrease the dynamic energy consumption. Typically, they save system energy by modifying the system voltage dynamically. Fig. 3 illustrates a representative DVFS algorithm [15].

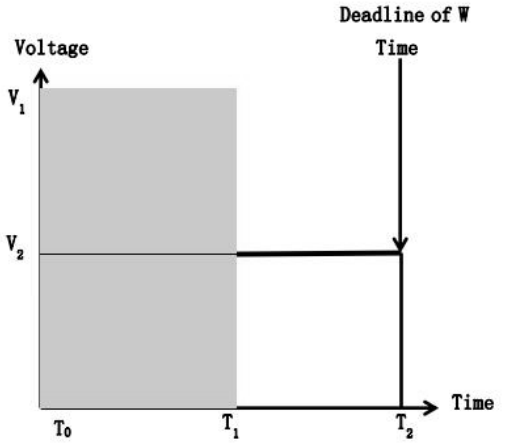


Fig. 3. A representative DVFS algorithm.

A DVFS algorithm must ensure the deadline of each scheduled task to satisfy the requirement of the service quality because the real-time task should be accomplished before its deadline [18]. To explain the DVFS algorithm in Fig. 3 clearly, let T_1 and T_2 denote the execution time of the task W performed at the voltage of V_1 and V_2, respectively [15]. Meanwhile, let F_1 and F_2 denote the frequency at the voltage of V_1 and V_2, respectively. Suppose that the relations between the variables are $V_2 = V_1/2$, $F_2 = F_1/2$ and $T_2 - T_1 = T_1 - T_0$. Depending on the energy model that will be discussed in the next section, if W is performed at the frequency F_1 and the voltage of V_1, the 75% dynamic energy can be reduced ideally compared with performing W at the frequency F_2 and the voltage of V_2. Although the execution time of W performed at the frequency F_1 and the voltage of V_1 is longer, W is still accomplished before its deadline and thus the service quality is also ensured.

An early presented DVFS algorithm is called the static DVFS which configures the voltage and the frequency according to the utilization of the taskset [19]. This algorithm does no longer modify the voltage and frequency when the scheduled tasks are preformed. Therefore, this algorithm fails to maximize the reduction of the dynamic energy in the case that the actual performing time varies in different periods. Another two typical DVFS algorithms are CC and LA [19]. CC first employs WCET to calculate the utilization of of the scheduled tasks, and then configures a high frequency at the start. After the tasks are performed,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

CC modifies the frequency dynamically from high to low according to the actual execution time (AET) based on the fact that AET is usually shorter than WCET. To save the dynamic energy, LA configures a low frequency when the tasks start. Furthermore, LA keeps a low frequency for the scheduled tasks until the coming deadlines when the frequency is increased to ensure the service quality.

More recently, DRA and AGR were presented [20]. DRA detects the early completions and accordingly reduces the frequency to decrease the dynamic energy consumption. To implement this scheme, DRA keeps a special data structure named α-queue. When a new task comes, DRA puts its worst case execution time into this data structure at S_{opt} frequency. Different tasks keep the same utilization at S_{opt} frequency. AGR can be viewed as an improved version of DRA. Specially, based on the assumption that the tasks existing in the data structure are accomplished before a new task comes, AGR transfers the processor time over the scheduled tasks in the data structure. In the case that the idle time is long enough, AGR can save the dynamic energy effectively. Other DVFS algorithms were presented to save the energy for multi-core scheduling depending on the system utilization and available cores.

According to the recent research that was conducted by Lin et al. [14], [21] who compared the performance of the typical DVFS algorithms for different scheduled tasksets, the performance of the DVFS algorithms greatly relies on the system state, for instance system utilization and dynamic slack. Therefore, none of the algorithms can obtain the best result for the dynamic energy consumption reduction in all cases.

2.2 Combined DVFS Algorithms

Since no single DVFS algorithm can obtain the best results for saving the dynamic energy in all cases, some combined DVFS algorithms have been presented to improve the robustness in different cases. Most of them are based on machine learning techniques. Specially, the basic idea of these algorithms is to learn a strategy from prior experiences and then to use the learned strategy to schedule the tasks for energy consumption reduction.

The most representative example of the combined DVFS algorithms is the hybrid DVFS scheme based on the reinforcement learning approach which was proposed by Islam and Lin [16]. Specially, they define the system state as dynamic slack and system utilization. For each system state, they use the Q-learning approach to learn a Q-value for every DVFS algorithm, which is stored into a Q-table. After learning the Q-table, the DVFS algorithm with a smallest Q-value is employed to modify the voltage and the frequency. This scheme achieved better results than any single DVFS algorithm in most cases. However, the Q-learning approach that is used in the hybrid DVFS scheme usually generates an overestimated Q-value, reducing the performance of the hybrid DVFS scheme. To overcome this drawback, Huang et al. [21] presented an improved hybrid DVFS scheme based the double Q-learning approach. In this scheme, the double Q-learning approach is used to learn the Q-table. Simulation results proved the better performance of this scheme than the hybrid DVFS scheme based on the Q-learning method. However, both the hybrid DVFS scheme and the improved hybrid DVFS scheme are only appropriate for the systems with discrete system states since they use a Q-table to store the relations between the Q-value of each DVFS algorithm and the system states. If the number of the system states are large or the system state is continuous, they have to expand the Q-table size at an exponential speed to improve the performance. To tackle this problem, Zhang et al. [15] presented a combined DVFS scheme based on the deep Q-learning model which also employs system utilization and dynamic slack as the system state. In this scheme, a stacked auto-encoder model is utilized to compute the Q-value of each DVFS algorithm for different system states instead of the Q-table. This scheme is particularly appropriate for the systems with continuous system states and obtained better performance. However, this scheme cannot obtain desirable results sometimes since it does not take the penalties of the future system states into account during the computation of the target Q-value for each DVFS algorithm, and thus it cannot obtain the optimal target Q-values in some cases. In addition, some algorithms by combining DVFS and DPM techniques based on online learning were also investigated in recent years [22].

3 System Models

3.1 Task Model

This work concentrates on energy-efficient scheduling for the systems with periodically real-time tasks for edge computing devices. A real-time task should be accomplished no later than its deadline to ensure the quality of service while a periodic task refers to the repeatedly executed one at regular periods.

In the system model used in our presented scheme, a taskset T is represented as [16]:

$$T = \{t_1(p_1, w_2), t_2(p_2, w_2), \ldots, t_n(p_n, w_n)\}. \quad (1)$$

where n denotes the number of tasks in T and t_i represents the ith task.

Specially, each task t is characterized by two parameters, i.e., period p and wcet w. Furthermore, each task t_i has a feature called utilization u_i which can be computed via:

$$u_i = w_i/p_i. \quad (2)$$

Similarly, T also has a utilization U computed via:

$$U = \sum\nolimits_{i=1}^{n} u_i. \quad (3)$$

Each task t_i generates a job in each period. In particular, we use t_i^j to represent the job generated in the jth period. Furthermore, we define a variable named hyperperiod for T as:

$$hyp = h \times LCM(p_1, p_2, \ldots, p_n). \quad (4)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

where LCM represents the least common multiple of all the periods int T and h represents the hth hyperperiod.

Each job in each hyperperiod is scheduled depending on its priority. Specially, this work uses the earliest deadline first (EDF) to compute each job's priority. In detail, the highest priority is set to the job with the earliest deadline [23].

3.2 Energy Model

In this part, we describe the energy model that is used to compute the energy consumption. Specially, we take the process based on CMOS for example to illustrate the energy model. Since this study concentrates on the reduction of the dynamic energy consumption, we introduce the dynamic energy consumption model in detail while the static energy consumption model can be referred in [15].

For a processor based on CMOS, the dynamic energy consumption P_d is computed via:

$$P_d = C_e \times V_{dd}^2 \times f, \tag{5}$$

where C_e, V_{dd}, and f are the switching capacity, the work voltage, and the work frequency, respectively.

After the voltage is set, the frequency is computed via:

$$f = \frac{(V_{dd} - V_{th})^\alpha}{L_d \times K_6}, \tag{6}$$

where V_{th} is the threshold voltage computed via:

$$V_{th} = V_{th1} - K_1 \times V_{dd} - K_2 \times V_{bs}. \tag{7}$$

Let P_s denote the static energy consumption that can be computed via:

$$P_{static} = V_{dd} \times I_{subn} + |V_{bs}| \times I_{jun}, \tag{8}$$

where I_{jun} denotes the reverse bias junction current and I_{subn} denotes the subthreshold leakage current that can be calculated via:

$$I_{subn} = K_3 \times e^{K_4 \times V_{dd}} \times e^{K_5 V_{bs}}. \tag{9}$$

$K_1, K_2, K_3, K_4, K_5, K_6, L_d, V_{th1}, V_{bs}$ and α denote the constants.

Specially, the utilized constants of the 70nm process technology in the dynamic energy consumption model are listed in Table 1 [15].

TABLE 1
Technique constants of the 70 nm technology.

Constant	Value	Constant	Value
K_1	0.063	V_{bs0}	0
K_2	0.153	α	1.5
K_3	5.38	V_{th1}	0.244
K_4	1.83	I_{jun}	4.8×10^{10}
K_5	4.19	C_e	0.43×10^9
K_6	5.26	L_d	37

Assuming that AT denotes the actual performing time of a task, it will consume the the total energy:

$$E_n = \theta \times P_d \times AT + \varphi \times P_s \times AT, \tag{10}$$

where θ and φ represent the weight constants.

4 Double Deep Q-learning Model

Given the taskset $T = \{t_1, t_2, \ldots, t_n\}$ and the available D-VFS algorithms $D = \{d_1, d_2, \ldots, d_m\}$, this study attempts to choose an optimal DVFS algorithm from D at the start of every hyperperiod to set the voltage and the frequency for the taskset T. The aim of this study is to minimize the energy consumption under the condition that each task can be accomplished before its deadline.

To this end, a double deep Q-learning model is designed to compute the Q-value for every available DVFS algorithm in D depending on the system state. **Specially, the Q-value denotes the expected average energy consumption of the DVFS algorithm in a hyperperiod.** Afterwards, the DVFS algorithm with the smallest Q-value will be chosen to set the voltage and the frequency for T.

Specially, the proposed double deep Q-learning model is illustrated in Fig. 4.

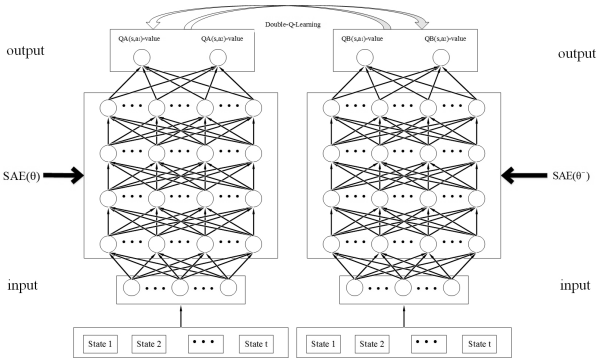


Fig. 4. Double deep Q-learning model.

From the architecture of the double deep Q-learning model presented in Fig. 4, the double deep Q-learning model includes two deep learning networks, called the generated Q-network (Q(s,a;θ) represented by SAE(θ)) and the target Q-network (Q(s,a;θ^-) represented by SAE(θ^-)). The generated Q-network is utilized to compute the Q-value for each available DVFS algorithm while the target Q-network aims to produce the target Q-values to train the parameters of the generated Q-network.

The generated Q-network inputs the system state characterized system utilization and dynamic slack to compute the Q-value for every available DVFS algorithm as output. Let $S = \{S_1, S_2, \ldots, S_n\}$ denote the state space with $S_i = \{su_i, ds_i\}$ where su and ds represent system utilization and dynamic slack, respectively. The system utilization is computed via:

$$su = \sum_{i=1}^{n} \frac{\omega_i}{p_i}. \tag{11}$$

The dynamic slack can be computed via:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

$$ds = 1 - \frac{Et_{hyp}}{\sum_{i=1}^{n} (\frac{hyp}{p_i} \times w_i)}, \tag{12}$$

where Et_{hyp} represents the sum of AET of each task in the hyperperiod hyp.

Et_{hyp} is computed via:

$$Et_{hyp} = \sum_{i=1}^{n} \sum_{j \in hyp} AET(\tau_i^j). \tag{13}$$

After the current system state $s = \{su, ds\}$ is fed to the generated Q-network Q(s,a;θ) with the parameters θ, the generated Q-network can compute the Q-value for each DVFS algorithm and then chooses a DVFS algorithm to set the voltage and the frequency. Afterwards, the next system state $s' = (su', ds')$ will be obtained. The target Q-network takes $s' = (su', ds')$ and the target Q-value for each DVFS algorithm as input and output, respectively.

Both the generated Q-network and the target Q-network employ the auto-encoder that is a typical artificial neural network as the basic module. An auto-encoder typically has an encoder and a decoder, as illustrated in Fig. 5 [24], [25].

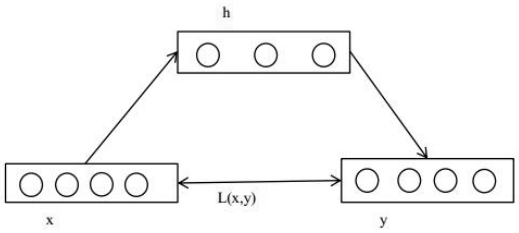


Fig. 5. Basic auto-encoder.

The encoder is utilized to project the input x to the hidden h via the activation function f:

$$h = f(W^{(1)}x + b^{(1)}). \tag{14}$$

In the deep Q-learning model with multiple DVFS algorithms (DQL-EES), the Sigmoid function is utilized as the activation function. Different from DQL-EES, we utilize the rectified linear units (ReLU) as the activation function instead of the Sigmoid function in this study due to the following reasons. **ReLU is more efficient than the Sigmoid function for gradient propagation. Furthermore, ReLU is more in accordance with the biological observation and it can address the problem of gradient vanishing effectively compared with the Sigmoid function. Actually, ReLU has achieved great success in computer vision and natural language processing [26], [27].**

Specially, ReLU is defined as:

$$f(x) = \max(x, 0). \tag{15}$$

The functional curve of ReLU is presented in Fig. 6.

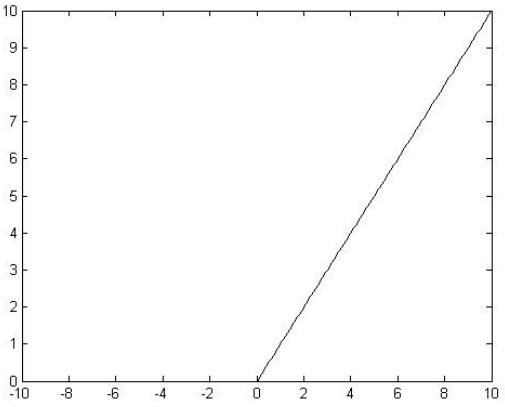


Fig. 6. Functional curve of ReLU.

The derivative of ReLU is:

$$f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}. \tag{16}$$

The decoder utilizes the decoding function g to reconstruct h to the output y:

$$y = g(W^{(2)}h + b^{(2)}), \tag{17}$$

where the rectified linear units (ReLU) or an identity function can be utilized as the decoding function.

To train the parameters $\theta = \{W^{(1)}, b^{(1)}; W^{(2)}, b^{(2)}\}$, a reconstruction objective $L(x, y)$ with a training object x is defined as:

$$L(x, y) = \frac{1}{2}||y - x||^2. \tag{18}$$

Furthermore, the global reconstruction objective $J(\theta)$ regarding m training objects is defined as:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}) + \frac{\lambda}{2} \sum_{l=1}^{2} W^{(l)2}, \tag{19}$$

where λ denotes the hyper parameter used to control the strength of the regularization.

Specially, the parameters are trained to minimize the reconstruction objective typically by the back-propagation strategy and the gradient descent algorithm, as described in Algorithm 1.

Similar to other deep Q-learning models, the explore-exploit dilemma is also faced in the proposed model. Since the ε-greedy strategy is an effective method to address this problem, the ε-greedy strategy is used to address the explore-exploit dilemma in this paper. Specially, at the beginning of each hyperperiod, a DVFS technique with probability ε randomly is selected and with probability $1 - \varepsilon$ to follow the current working policy.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

Algorithm 1: Auto-encoder Training.

Input: $\{(X^{(i)}, Y^{(i)})\}_{i=1}^{m}$, η, $threshold$
Output: $\theta = \{W^{(1)}, b^{(1)}; W^{(2)}, b^{(2)}\}$

for $iteration = 1, 2, ..., iterater_{max}$ **do**
  for $example = 1, 2, ..., N$ **do**
    for $j = 1, 2, \ldots, m$ **do**
      // Compute the hidden layer represented by $a_j^{(2)}$;
      $a_j^{(2)} = f(\sum_{i=1}^{n} W_{ji}^{(1)} + b_i^{(1)})$;
      Let $z_j^{(2)} = \sum_{i=1}^{n} W_{ji}^{(1)} + b_i^{(1)}$;
    for $i = 1, 2, \ldots, n$ **do**
      // Compute the output layer represented by $a_i^{(3)}$;
      $a_i^{(3)} = f(\sum_{j=1}^{m} W_{ij}^{(2)} a_j^{(2)} + b_i^{(2)})$;
      Let $z_i^{(3)} = \sum_{j=1}^{m} W_{ij}^{(2)} a_j^{(2)} + b_i^{(2)}$;
    if $J_{TAE}(\theta) > threshold$ **then**
      for $i = 1, 2, \ldots, n$ **do**
        // Compute the residual of output layer represented by $\sigma_i^{(3)}$;
        $\sigma_i^{(3)} = (a_i^{(3)} - x_i) \cdot f'(z_i^{(3)})$;
      for $j = 1, 2, \ldots, m$ **do**
        // Compute the residual of hidden layer represented by $\sigma_j^{(2)}$;
        $\sigma_j^{(2)} = (\sum_{i=1}^{n} W_{ij}^{(2)} \sigma_i^{(3)}) \cdot f'(z_j^{(2)})$;
      for $i = 1, 2, \ldots, n$ **do**
        // Compute the partial derivative $\Delta b_i^{(2)}$;
        $\Delta b_i^{(2)} = \Delta b_i^{(2)} + \sigma_i^{(3)}$;
        for $j = 1, 2, \ldots, m$ **do**
          // Compute the partial derivative $\Delta w_{ij}^{(2)}$;
          $\Delta w_{ij}^{(2)} = \Delta w_{ij}^{(2)} + a_j^{(2)} \cdot \sigma_i^{(3)}$;
      for $j = 1, 2, \ldots, m$ **do**
        // Compute the partial derivative $\Delta b_j^{(1)}$;
        $\Delta b_j^{(1)} = \Delta b_j^{(1)} + \sigma_j^{(2)}$;
        for $i = 1, 2, \ldots, n$ **do**
          // Compute the partial derivative $\Delta w_{ji}^{(1)}$;
          $\Delta w_{ji}^{(1)} = \Delta w_{ji}^{(1)} + x_i \cdot \sigma_j^{(2)}$;
      // Update parameters using the gradient descent algorithm;
      $W = W - \eta \times (\frac{1}{N} \Delta w)$;
      $b = b - \eta \times (\frac{1}{N} \Delta b)$;

5 Learning Algorithm for Double Deep Q-learning Model

In this section, we design a learning algorithm to train the double deep Q-learning model.

In the deep Q-learning model with multiple DVFS algorithms (DQL-EES), the Q-value of each DVFS algorithm produced by the Q-learning model is utilized as the target Q-value to train the parameters. Let $Q(s_t, a_t)$ and $Q(s_t, a_t; \theta)$ denote the Q-values of the DVFS algorithm a_t at the system state s_t produced by the Q-learning model and the deep Q-learning model, respectively. Specially, $Q(s_t, a_t)$ can be computed via:

$$\begin{aligned} &Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \\ &\times [p_t(s_t, a_t) - Q(s_t, a_t)] \end{aligned}, \tag{20}$$

where $p_t(s_t, a_t)$ represents the penalty when using the DVFS algorithm a_t to set the voltage and the frequency at the system state s_t.

In DQL-EES, $p_t(s_t, a_t)$ is computed via [15]:

$$p_t(s_t, a_t) = \frac{En(s_t, a_t)}{Et_{hyp}}, \tag{21}$$

where $En(s_t, a_t)$ represents the energy consumption when using the DVFS algorithm a_t to set the voltage and the frequency at the system state s_t. Therefore, the penalty denotes the average energy consumption consumption in the corresponding hyperperiod.

Furthermore, DQL-EES trains the parameters by minimizing the following objective:

$$\begin{aligned} &L(\theta) = \sum_{i=1}^{m} [Q(s_i, a_i, \theta) - (Q(s_i, a_i) \\ &+ \alpha_i \times [p_i(s_i, a_i) - Q(s_i, a_i)])]^2 \end{aligned}, \tag{22}$$

where m denotes the number of the transitions selected from the replay experience randomly.

From Eq.(18), DQL-EES does not take the penalties of the future system states into account during the computation of the target Q-value for each DVFS algorithm, and thus it cannot obtain the optimal target Q-values in some cases. In this study, we utilize the target Q-network instead of the Q-learning model to generate the target Q-values to train the parameters.

Our proposed learning scheme is illustrated in Fig. 7.

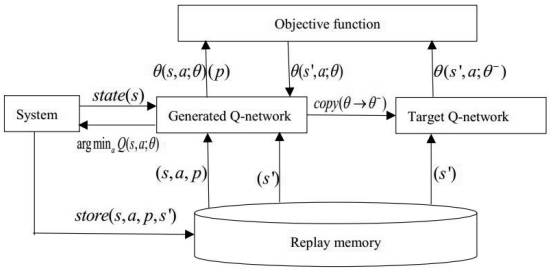


Fig. 7. The learning scheme for training the parameters.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

Our proposed learning scheme is based on experience replay which is utilized to store the transitions. Specially, a transition is represented by a quadruple $e = (s, a, p(s,a), s')$ where s denotes the current system state, a denotes the chosen DVFS algorithm, $p(s,a)$ denotes the corresponding penalty, and s' denotes the next system state altered from s with using a to set the voltage and the frequency. In detail, at the start of the tth hyperperiod, the system state s_t is observed and a DVFS algorithm a_t is selected to set the voltage and the frequency for scheduling the tasks. After the hpyerperiod is over, the new system state s_{t+1} can be observed and the penalty $p_t(s_t, a_t)$ can be computed. Thus, we can obtain a transition $e_t = (s_t, a_t, p_t(s_t, a_t), s_{t+1})$ and store it into the experience replay. Before the parameters are trained, we must collect enough transitions to provide the training samples. Specially, the experience replay with N transitions can be represented as $E = \{e_1, e_2, \ldots, e_N\}$.

To train the parameters of the presented double deep Q-learning model, we randomly select m transitions from E as the training samples. For each training sample $e = (s, a, p(s,a), s')$, we input the system state s into the generated Q-network to compute the Q-value $Q(s, a; \theta)$ for the DVFS algorithm a. Afterwards, we input the system state s' into the generated Q-network to compute the Q-value $Q(s', a; \theta)$ for each DVFS algorithm. Finally, we input the system state s' into the target Q-network to compute the Q-value $Q(s', a; \theta^-)$ for each DVFS algorithm. Depending on the basic idea of the double deep Q-learning, the target Q-value Y with this training sample can be defined as:

$$Y = p(s,a) + \gamma Q(s', \arg\min_a Q(s', a; \theta), \theta^-). \quad (23)$$

Furthermore, the objective function $L(\theta)$ with m training samples is defined to learn the parameters as:

$$L(\theta) = \frac{1}{m} \sum_{i=1}^{m} (Q(s_i, a_i; \theta) - Y_i)^2. \quad (24)$$

Specially, the parameters are trained to minimize the objective function. Every time the parameters θ is updated for K times, they are copied to θ^-. Once the objective function is fixed, the parameters θ can be trained by combining the back-propagation strategy and the gradient descent algorithm.

Overall, our presented learning scheme for training the parameters of the double deep Q-learning model is illustrated in Algorithm 2.

From Algorithm 2, collecting transitions and executing back-propagation to train the parameters account for the computational complexity. Since collecting one transition requires $O(1)$ computational complexity, the total computational complexity for collecting N transitions into the replay memory is $O(N)$. Let p and q denote the number of layers and the maximum number of units in each layer, respectively. Training parameters with back-propagation and gradient descent requires the computational complexity of $O(mpqi)$ where m and i denote the number of transitions randomly sampled from the replay memory and the number of iterations, respectively. Furthermore, the replay memory and the parameters of the double deep Q-learning model dominate the storage complexity. Specially, storing N transitions needs the about space complexity of $O(N)$ while the parameters need the about space complexity of $O(pq)$.

Algorithm 2: Learning scheme for training the parameters of the double deep Q-learning model.

```
Collect enough transitions to store the experience replay E;
Randomly select m transitions from E as the training samples;
for i = 1, ..., m do
    Input the system state s into the generated Q-network;
    Compute the Q-value Q(s, a; θ) ;
    Input the system state s' into the generated Q-network;
    Compute the Q-value Q(s', a; θ) ;
    Input the system state s' into the target Q-network ;
    Compute the Q-value Q(s', a; θ⁻) for each DVFS algorithm;
    Compute the target Q-value:;
    Y = p(s, a) + γQ(s', arg max_a(s', a; θ), θ⁻)
Perform the back-propagation algorithm to train θ;
```

6 Simulation and Experiment

In this section, we compare our presented model (DDQ-EES) with the deep Q-learning model with combined DVFS algorithms (DDQ-EES) for energy saving on EdgeCloudSim which is a new environment for performance evaluation of edge computing applications [28]. For a fair comparison, DDQ-EES combines cycle-conserving, look-ahead and dynamic reclaiming algorithm since they are used in DQL-EES. Furthermore, we run DQL-EES and DDQ-EES to schedule the a simulation taskset and an experiment taskset on EdgeCloudSim.

Since the features of every task, including the number of jobs, period, and wcet in the simulation taskset and the experimental taskset, are different from other tasks, each group has a distinct system state. In the following parts, we report the energy consumption that is normalized with respect to the highest one to compare the performance of DQL-EES and DDQ-EES, which is a commonly utilized methodology in other studies.

6.1 Results on the Simulation Taskset

The details of the simulation taskset with twenty tasks are listed in the literature [15]. In this study, we utilize this taskset to produce four different groups in the same way with DQL-EES.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

We first compare the performance of DQL-EES and DDQL-EES in terms of energy consumption. Since the number of hidden layers and the number of hidden neurons in each hidden layer have a great impact for two models, we built multiple deep Q-learning models and double deep Q-learning models, each has the distinct number of hidden neurons. In the simulations, each model is trained for five times and we present the average energy consumption. Furthermore, we utilize DQL-n and DDQ-n to denote the deep Q-learning model and the double deep Q-learning model with n hidden layers, respectively, to present the results clearly.

TABLE 2
Average energy consumption on the first group with 4 tasks

Model	4	8	12	16
DQL-1	0.9755	0.9732	0.9718	0.9717
DDQ-1	0.9583	0.9519	0.9487	0.9499
DQL-2	0.9319	0.9294	0.9277	0.9278
DDQ-2	0.9153	0.9102	0.9087	0.9069
DQL-3	1	0.9982	0.9954	0.9951
DDQ-3	0.9751	0.9722	0.9693	0.9701

TABLE 3
Average energy consumption on the second group with 6 tasks

Model	4	8	12	16
DQL-1	0.9903	0.9891	0.9864	0.9859
DDQ-1	0.9673	0.9723	0.9589	0.9516
DQL-2	0.9278	0.9251	0.9219	0.9219
DDQ-2	0.9101	0.9027	0.9009	0.8996
DQL-3	1	0.9969	0.9957	0.9958
DDQ-3	0.9798	0.9722	0.9709	0.9683

TABLE 4
Average energy consumption on the third group with 3 tasks

Model	4	8	12	16
DQL-1	0.9487	0.9452	0.9431	0.9429
DDQ-1	0.9269	0.9206	0.9197	0.9201
DQL-2	0.8943	0.8941	0.8902	0.8900
DDQ-2	0.8763	0.8709	0.8688	0.8689
DQL-3	1	0.9942	0.9926	0.9923
DDQ-3	0.9764	0.9731	0.9699	0.9691

We can draw three remarkable conclusions from Tables 2-5. First, when the number of the hidden layers is fixed in both DQL-EES and DDQ-EES, adding the hidden neurons can reduce more energy consumption in most cases. For instance, when the double deep Q-learning model has two hidden layers, the average energy consumption reduces from 0.9153 to 0.9087 with the increasing number of the hidden neurons from 4 to 8. Such results demonstrate that the performance of both DQL-EES and DDQ-EES could be improved by increasing hidden neurons. Second, both

TABLE 5
Average energy consumption on the fourth group with 7 tasks

Model	4	8	12	16
DQL-1	0.9531	0.9501	0.9485	0.9482
DDQ-1	0.9344	0.9317	0.9299	0.9285
DQL-2	0.9296	0.9282	0.9253	0.9254
DDQ-2	0.9091	0.9074	0.9019	0.9023
DQL-3	1	0.9984	0.9968	0.9967
DDQ-3	0.9496	0.9412	0.9327	0.9308

DQL-EES and DDQ-EES with two hidden layers reduce the energy consumption significantly compared to two models with only one hidden layer, indicating that their performance could be improved by increasing hidden layers. However, when DQL-EES has more than two hidden layers, its performance becomes worse possibly due to the overfitting. In contrast, DDQ-EES with three hidden layers can still achieve a good performance for reducing the energy consumption, implying that DDQ-EES performs better than DQL-EES. Finally, DQL-EES consumes less energy than DQL-EES for four groups of tasks. Therefore, DDQ-EES is more effective than DQL-EES to reduce the energy consumption for the system with multiple periodically real-time tasks.

Table 6 lists the running time of DQL-EES and DDQ-EES for learning their parameters on the simulation taskset.

TABLE 6
Average training time on the simulation taskset

Model	Group 1	Group 2	Group 3	Group 4
DQL-1	0.79	0.72	0.82	0.69
DDQ-1	0.67	0.71	0.75	0.62
DQL-2	0.81	0.79	0.94	0.81
DDQ-2	0.68	0.76	0.81	0.69
DQL-3	1	1	1	1
DDQ-3	0.84	0.86	0.87	0.72

Table 6 implies two remarkable conclusions. First, with the growing number of the hidden layers of both DQL-EES and DDQ-EES, the running time of two models increses since more parameters need to be trained. For example, for the double deep Q-learning model, the running time increases from 0.71 to 0.86 on the second group of tasks for training the parameters when the size of the hidden layers is added from 1 to 3. Second, DQL-EES is more time-consuming than DDQ-EES for training the parameters since DQL-EES needs to pre-train the parameters which is very time-consuming. For example, DDQ-EES saves 28% time compared with DQL-EES when they have three hidden layers for scheduling the fourth group. Such observation proves that DDQ-EES is more efficient than DQL-EES.

6.2 Results on the Experimental Taskset

We use one MiBench as the experimental taskset which includes a set of commercially representative embedded programs [29]. In the experiment, we use this taskset to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

generate two different groups, i.e., one group with 6 tasks and the other with 4 tasks, and then we schedule them on EdgeCloudSim.

We still compare the performance of DQL-EES and DDQL-EES with the different number of hidden layers and hidden neurons in terms of energy consumption, as presented in Tables 7 and 8.

TABLE 7
Average energy consumption on the first group

Model	4	8	12	16
DQL-1	0.9954	0.9916	0.9819	0.9801
DDQ-1	0.9792	0.9729	0.9659	0.9651
DQL-2	0.9802	0.9716	0.9621	0.9665
DDQ-2	0.9613	0.9524	0.9491	0.9413
DQL-3	1	0.9936	0.9901	0.9758
DDQ-3	0.9792	0.9621	0.9609	0.9513

TABLE 8
Average energy consumption on the second group

Model	4	8	12	16
DQL-1	0.9896	0.9821	0.9803	0.9779
DDQ-1	0.9657	0.9622	0.9598	0.9601
DQL-2	0.9359	0.9302	0.9263	0.9298
DDQ-2	0.9126	0.9117	0.9103	0.9119
DQL-3	1	0.9797	0.9631	0.9636
DDQ-3	0.9801	0.9581	0.9413	0.9496

From Table 7 and Table 8, with the growing number of hidden units, the energy consumption reduces gradually. However, the model with 12 hidden units consumes almost the same energy with the model with 16 hidden units, so 12 hidden units are enough for this experimental taskset to reduce the dynamic energy consumption. Furthermore, DQL-EES consumes less energy than DQL-EES for two groups of tasks. **For example, when two deep models have 3 hidden layers and each hidden layer has 12 neurons, the energy consumption on the first group of DDQ and DQL is 0.9609 and 0.9901 while the energy consumption on the second group of the two models is 0.9413 and 0.9631, respectively.** Therefore, DDQ-EES is more effective than DQL-EES for energy-efficiency edge computing.

Table 9 illustrates the running time of DQL-EES and DDQ-EES for learning their parameters on the experimental taskset.

TABLE 9
Average training time on the experimental taskset

Model	Group 1	Group 2
DQL-1	0.78	0.72
DDQ-1	0.52	0.51
DQL-2	0.89	0.84
DDQ-2	0.65	0.62
DQL-3	1	1
DDQ-3	0.79	0.76

Table 9 demonstrates that DQL-EES is more time-consuming than DDQ-EES for training the parameters since DQL-EES needs to pre-train the parameters which is very time-consuming. Specially, DDQ-EES saves average 22% time compared with DQL-EES when they have three hidden layers for scheduling the fourth group. Such observation proves that DDQ-EES is more efficient than DQL-EES.

7 Conclusion

In this study, we propose a double deep Q-learning model to reduce the energy consumption for the systems with periodically real-time tasks in edge computing devices. There are two advantages of the presented model. The first advantage is to utilize the rectified linear units (ReLU) as the activation function for improving the training efficiency. Second, a target Q-network is designed instead of the Q-learning model to generate more objective target Q-values that are utilized to train the parameters more effectively. Two attractive conclusions are drawn from the simulation and experimental results on EdgeCloudSim: (1) the presented DDQ-EES model can save average $2\% - 2.4\%$ energy than QQL-EES; (2) DDQ-EES is more efficient than DQL-EES for training the parameters; indicating the potential of the proposed model for energy-efficient edge scheduling. **In this study, we did not focus on the explore-exploit tradeoff deeply. Actually, an effective explore-exploit strategy plays a significant role in a deep reinforcement learning model, so we will apply the advanced explore-exploit strategies to the presented model to improve its performance for edge energy saving further in the future work. Furthermore, we will investigate the use of tensor decompositions and tensor-train networks [30], [31], [32], [33] to improve the training efficiency of the presented double deep Q-learning model in the future work.**

References

[1] M. Z. A. Bhuiyan, J. Wu, G. Wang, Z. Chen, J. Chen, and T. Wang, "Quality Guaranteed and Event-Senstive Data Collection and Monitoring in Wireless Vibration Sensor Networks," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 572-583, 2017.

[2] D. Zhang, L. T. Yang, M. Chen, S. Zhao, M. Guo, and Y. Zhang, "Real-Time Locating Systems Using Active RFID for Internet of Things," *IEEE Systems Journal*, vol. 10, no. 3, pp. 1226-1235, 2016.

[3] X. Liu, H. Zhu, J. Ma, Q. Li, and J. Xiong, "Efficient Attribute Based Sequential Aggregate Signature for Wireless Sensor Networks," *International Journal of Sensor Networks*, vol. 16, no. 3, pp. 172-184, 2014.

[4] H. Zhou, Victor C. M. Leung, C. Zhu, S. Xu, and J. Fan, "Predicting Temporal Social Contact Patterns for Data Forwarding in Opportunistic Mobile Networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 11, pp. 10372-10383, 2017.

[5] Z. Zhou, M. Dong, K. Ota, G. Wang, and L. T. Yang, "Energy-Efficient Resource Allocation for D2D Communications Underlaying Cloud-RAN-Based LTE-A Networks," *IEEE Internet of Things Journal*, vol. 3, no. 3, pp. 428-438, 2016.

[6] P. Li, Z. Chen, L. T. Yang, Q. Zhang, and M. J. Deen, "Deep Convolutional Computation Model for Feature Learning on Big Data in Internet of Things," *IEEE Transactions on Industrial Informatics*, 2017, DOI: 10.1109/TII.2017.2739340.

[7] D. He, C. Chen, S. Chan, J. Bu, and L. T. Yang, "Security Analysis and Improvement of a Secure and Distributed Reprogramming Protocol for Wireless Sensor Networks," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 11, pp. 5348-5354, 2013.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

[8] P. Li, Z. Chen, L. T. Yang, L. Zhao, and Q. Zhang, "A Privacy-Preserving High-order Neuro-fuzzy c-Means Algorithm with Cloud Computing," *Neurocomputing*, vol. 256, pp. 82-89, 2017.

[9] Z. Shao, M. Wang, Y. Chen, C. Xue, M. Qiu, L. T. Yang, and E. H. -M. Sha, "Real-Time Dynamic Voltage Loop Scheduling for Multi-Core Embedded Systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 5, pp. 445-449, 2007.

[10] X. Deng, Z. Tang, L. T. Yang, M. Lin, and B. Wang, "Confident Information Coverage Hole Healing in Hybrid Industrial Wireless Sensor Networks," *IEEE Transactions on Industrial Informatics*, 2017, DOI: 10.1109/TII.2017.2764038.

[11] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "PPHOPCM: Privacy-preserving High-order Possibilistic c-Means Algorithm for Big Data Clustering with Cloud Computing," *IEEE Transactions on Big Data*, 2017, DOI: 10.1109/TBDATA.2017.2701816.

[12] X. Liu, K. R. Choo, R. H. Deng, R. Lu, and J. Weng, "Efficient and Privacy-Preserving Outsourced Calculation of Rational Numbers," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 27-39, 2018.

[13] M. Dong, H. Li, K. Ota, L. T. Yang, and H. Zhu, "Multicloud-based Evacuation Services for Emergency Management," *IEEE Cloud Computing*, vol. 1, no. 4, pp. 50-59, 2014.

[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, 2016.

[15] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, and P. Li, "Energy-Efficient Scheduling for Real-Time Systems Based on Deep Q-Learning Model," *IEEE Transactions on Sustainable Computing*, 2017, DOI: 10.1109/TSUSC.2017.2743704.

[16] F. Islam and M. Lin, "Hybrid DVFS Scheduling for Real-Time Systems Based on Reinforcement Learning," *IEEE Systems Journal*, vol. 11, no. 2, pp. 931-940, 2017.

[17] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A Survey on Deep Learning for Big Data," *Information Fusion*, vol. 42, pp. 146-157, 2018.

[18] X. Lin, Y. Wang, N. Chang, and M. Pedram, "Concurrent Task Scheduling and Dynamic Voltage and Frequency Scaling in a Real-Time Embedded System With Energy Harvesting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 11, pp. 1890-1902, 2016.

[19] P. Pillai and K. G. Shin, "Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems," in *Proceedings of the 8th ACM Symposium on Operating Systems Principles*, 2001, pp. 89-102.

[20] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Time-efficient Power-aware Scheduling for Periodic Real-time Tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584-600, 2004.

[21] H. Huang, M. Lin, and Q. Zhang, "Hybrid DVFS Scheduling for Real-Time Systems Based on Reinforcement Learning," in *Proceedings of IEEE International Conference on Green Computing and Communications Conference*, 2017, pp. 522-529.

[22] M. Bhatti, C. Belleudy, and M. Auguin, "Hybrid Power Management in Real Time Embedded Systems: An Interplay of DVFS and DPM Techniques," *Real-Time Systems*, vol. 47, no. 2, pp. 143-162, 2011.

[23] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Time-efficient Power-aware Scheduling for Periodic Real-time Tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584-600, 2004.

[24] Q. Zhang, L. T. Yang, Z. Yan, Z. Chen, and P. Li, "An Efficient Deep Learning Model to Predict Cloud Workload for Industry Informatics," *IEEE Transactions on Industrial Informatics*, 2018, DOI: 10.1109/TII.2018.2808910.

[25] Q. Zhang, L. T. Yang, Z. Chen, P. Li, and M. J. Deen, "Privacy-preserving Double-projection Deep Computation Model with Crowdsourcing on Cloud for Big Data Feature Learning," *IEEE Internet of Things Journal*, 2017, DOI: 10.1109/JIOT.2017.2732735.

[26] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315-323.

[27] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units," in *Proceedings of International Conference on Machine Learning*, 2016, pp. 2217-2225.

[28] C. Sonmez, A. Ozgovde, and C. Ersoy, "EdgeCloudSim: An Environment for Performance Evaluation of Edge Computing Systems," in *Proceedings of International Conference on Fog and Mobile Edge Computing*, 2017, pp. 39-44.

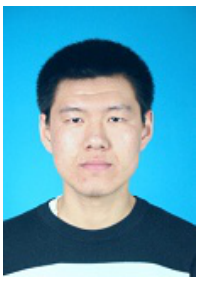
[29] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in *Proceedings of IEEE International Workshop on Workload Characterization*, 2001, pp. 3-14.

[30] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "An Improved Deep Computation Model Based on Canonical Polyadic Decomposition," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2017, DOI: 10.1109/TSMC.2017.2701797.

[31] T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Review*, vol. 51, no. 3, pp. 455-500, 2009.

[32] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A Tensor-train Deep Computation Model for Industry Informatics Big Data Feature Learning," *IEEE Transactions on Industrial Informatics*, 2018, DOI: 10.1109/TII.2018.2791423.

[33] I. V. Oseledets, "Tensor-Train Decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295-2317, 2011.



Qingchen Zhang received the B.Eng. degree in Southwest University, China, and the Ph.D. degree in Dalian University of Technology, China. He is currently a postdoc fellow in St. Francis Xavier University. His research interests include Big Data and Deep Learning.

Man Lin received the B.E. degree in computer science and technology from Tsinghua University,Beijing, China, and the Ph.D. degree from Linköping University, Linköping, Sweden. She is currently a Professor of computer science with St. Francis Xavier University, Antigonish, NS, Canada. Her research interests include realtime and embedded system scheduling, power-aware computing, parallel algorithms, and optimization algorithms.

Laurence T. Yang received the B.E. degree at Tsinghua University, China, and the Ph.D degree at University of Victoria, Canada. He is currently a professor at University of Electronic Science and Technology of China and St. Francis Xavier University, Canada. His research has been supported by the National Sciences and Engineering Research Council, and the Canada Foundation for Innovation. His research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing, big data and cyber-physical-social systems.

Zhikui Chen received the B.Eng. degree in Chongqing Normal University, China, and the Ph.D. degree in Chongqing University, China. He is currently a professor in Dalian University of Technology, China. His research interests include Internet of Things and Big Data Processing.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867482, IEEE Transactions on Services Computing

Samee U. Khan received the B.Eng. degree in Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Pakistan, and the Ph.D. degree in University of Texas, USA. Currently, he is an Associate Professor of Electrical and Computer Engineering in North Dakota State University, USA. His research interests include optimization, robustness, and security of cloud, grid, cluster and big data computing, social networks, wired and wireless networks, power systems, smart grids, and optical networks. He is an ACM Distinguished Speaker, an IEEE Distinguished Lecturer, a Fellow of the Institution of Engineering and Technology (IET, formerly IEE), and a Fellow of the British Computer Society (BCS).

Peng Li received the B.Eng. degree in Dezhou University, China. He is a Ph.D. student in Dalian University of Technology, China. His research interests include Deep Learning and Big Data.