Renewable Energy-Aware Big Data Analytics in Geo-distributed Data Centers with Reinforcement Learning

Chenhan Xu, Kun Wang, Senior Member, IEEE, Peng Li, Member, IEEE, Rui Xia, Song Guo, Senior Member, IEEE, and Minyi Guo, Fellow, IEEE

Abstract—In the age of big data, companies tend to deploy their services in data centers rather than their own servers. The demands of big data analytics grow significantly, which leads to an extremely high electricity consumption at data centers. In this paper, we investigate the cost minimization problem of big data analytics on geo-distributed data centers connected to renewable energy sources with unpredictable capacity. To solve this problem, we propose a Reinforcement Learning (RL) based job scheduling algorithm by combining RL with neural network (NN). Moreover, two techniques are developed to enhance the performance of our proposal. Specifically, Random Pool Sampling (RPS) is proposed to retrain the NN via accumulated training data, and a novel Unidirectional Bridge Network (UBN) structure is designed for further enhancing the training speed by using the historical knowledge stored in the trained NN. Experiment results on real Google cluster traces and electricity price from Energy Information Administration show that our approach is able to reduce the data centers’ cost significantly compared with other benchmark algorithms.

1 INTRODUCTION

Internet services, such as web-mail, distance education, searching and online chatting, have gained great popularity in recent years. In order to achieve global service coverage and high availability, service providers have utilized multiple geo-distributed data centers to conduct streaming analytics on big data continuously collected from users [1], [2]. However, running such kinds of big data jobs on data centers consumes overwhelming amount of energy, which causes a serious burden to the environment and the economy. The statistics report [3] shows that the energy consumed by data centers accounts for 1.3% of total energy consumption of the world in 2010.

There have been emphasizing research efforts on energy issues of geo-distributed data centers. Earlier studies focus on energy efficiency by employing various energy-saving techniques, e.g., dynamic CPU voltage adjusting [4] and resizing [5]. Recent work [6]–[8] starts to exploit renewable energy, such as wind and solar, to power data centers so that the energy consumption from traditional power grid can be significantly reduced. However, how to use renewable energy to decrease the cost of big data analytics is still an open challenge [9]–[11].

In this paper, we investigate a renewable energy-aware job scheduling issue in geo-distributed data centers based on streaming big data analytics. Particularly, we take a set of streaming big data jobs into consideration, each of which runs on a cluster of virtual machines accommodated in several geo-distributed data centers connected to both traditional power grid and renewable energy sources with unpredictable capacity [12]. When more renewable energy has been generated at a data center due to favorable weather conditions, migrating big data jobs to this data center could decrease energy consumption from power grid under an incurred migration overhead. This cost might be high when migration meets network traffic congestion. Our proposal is designed for minimizing the total cost of energy consumption from grid and job migration without any knowledge of future renewable energy generation. To solve this challenging problem, we propose a novel job scheduling algorithm based on reinforcement learning (RL) [13] that can approximate the optimal solution by iteratively learning the feedback from historical job scheduling decisions (i.e., job locations in different time intervals), which are also referred to as actions. The learning process of RL consists of a sequence of actions and the corresponding rewards. In each iteration, RL maintains a value function to evaluate the expected effect of taking different actions. When the action selected by value function is applied, RL observes the state that appears thereafter, e.g., the current generated renewable energy and the data center load, the reward associated with this state that can be used to refine its value function. Although RL is a promising approach, the following challenges need to
be addressed on how to quickly approximate the optimal solution for streaming big data analytics.

- **Value function design.** Designing a value function to evaluate different job scheduling actions is difficult because the system state in our problem depends on many factors, such as generated renewable energy (weather condition), resource utilization and network congestion. Furthermore, since there is no priori knowledge of future renewable energy generation, it is difficult to evaluate the cost saving of current scheduling decision.

- **Sub-optimal and slow convergence.** The traditional RL learns knowledge from the rewards of previous scheduling decisions using temporal differences [15]. Our experiments on real data sets reveal that this method fails to guarantee quick convergence to the optimal solution that can be obtained when the knowledge of future dynamics is available.

Different from existing work striving for value function design, we propose to use neural network (NN) to evaluate different job scheduling actions that will exhibit high accuracy as shown later in our experiments. NN is composed of a number of interconnected processing elements, also called neurons, organized into several layers. Given the current state as input, it can learn how to calculate an output (i.e., expected cost) based on a set of training data. Due to the strong learning capability of NN, we integrate it into the RL framework, such that we can adjust job scheduling to approximate the optimal solution with minimum cost [16]. To further improve training accuracy, we propose a novel random pool sampling (RPS) method to improve training data quality by caching historical data and periodically retrain the NN based on them. A unidirectional bridge network (UBN) structure is designed to reuse some neurons of NN constructed in previous iterations, such that the training process can be accelerated. We summarize the contributions in this paper as follows:

- We are the first to study the streaming big data analytics with renewable energy by formulating a big data job scheduling problem in geo-distributed data centers. Our proposal is designed for minimizing the total cost of energy consumption from traditional power grid and job migration among data centers.

- We develop an RL-based algorithm to deal with the job scheduling issue, without any assumption of future renewable energy generation and job migration cost. NN is applied to simplify the value function design in the traditional RL algorithm. This paper is the first to embed NN into RL for its value function design in the studied problem.

- To improve the performance of our proposed scheme, two techniques have been developed. In order to increase the accuracy of NN, we propose an RPS approach that periodically retrains the NN using accumulated training data with improved quality. We also design a novel unidirectional bridge network (UBN) structure to further enhance the training speed of our proposed algorithm by using historical knowledge stored in the trained NN. The experimental results show that these techniques can improve the cost saving and convergence speed by 20% and 200%, respectively.

- Extensive simulations are performed based on various data traces of real world workload, renewable energy, and grid electricity price for assessing the performance of our proposed algorithm. Our proposal has superior performance than benchmark approaches according to numerical results analysis.

We organize the rest of this paper as follows. Section II elaborates the system model and problem formulation. An RL based job scheduling algorithm is presented in Section. In Section IV, we enhance our proposed algorithm using a random pool sampling approach and a new NN structure. The results of extensive experiments are shown in Section V. Section VI surveys the related work. Section VII gives the conclusion of our paper finally.

## 2 SYSTEM MODEL AND PROBLEM FORMULATION

We consider a set \( D = \{d_1, d_2, \ldots, d_m\} \) of data centers with renewable energy sources distributed in different geographical regions, which is shown in Fig. 1. Similar with the model adopted in [17], streaming data from users first arrive at a number of gateways, which then forward them to different data centers. Particularly, we consider a set \( J = \{j_1, j_2, \ldots, j_n\} \) of big data jobs, each of which runs on a cluster of virtual machines. The data arriving rate of job \( j_k \) is denoted by \( c_k \) that would fluctuate over time. When a large amount of data arrives, the increased resource utilization of virtual clusters will lead to higher energy consumption.

Based on this observation, we consider a discrete-time power model of \( T \) time intervals, in which the total energy consumption of data center \( d_i \) at time interval \( t \) can be calculated by

\[
u_{i,t} = \int_{t_0}^{t} \left[ \sum_{d_i \in D} P_i(t) \right] dt, \quad (1)\]

where

\[
P_i(t) = p_{i}^{\text{idle}} + \left( p_{i}^{\text{full}} - p_{i}^{\text{idle}} \right) \left( 2r_{c}^{\varepsilon}(t) - r_{c}^{\varepsilon}(t)^{1.4} \right). \quad (2)\]

In (2), \( r_{c}^{\varepsilon}(t) \) represents the utilized CPU resource of data center \( d_i \) at time interval \( t \). \( p_{i}^{\text{idle}} \) and \( p_{i}^{\text{full}} \) denotes the power of \( d_i \) when the utilization of it’s CPU is at 0% and 100%, respectively [18].

In our model, the energy supply of a data center includes green renewable energy and brown energy from grid. The
policy of energy using is to give high priority to deploying the green energy, where \( e_{i,t} \) is denoted as the amount of generated green energy at time interval \( t \). \( \rho_{i,t}^{\text{brown}} \) and \( \rho_{i,t}^{\text{green}} \) represent the price of brown and green energy at time interval \( t \), respectively. Energy cost at time interval \( t \) can be calculated as follows:

\[
C_{i}^{\text{energy}} = \max \left( u_{i,t} - e_{i,t}, 0 \right) \rho_{i,t}^{\text{brown}} + \min \left( u_{i,t}, e_{i,t} \right) \rho_{i,t}^{\text{green}}. \tag{3}
\]

Moving a job to a different data center involves the migration of virtual clusters and redirection of data forwarding policies at gateways. Let \( d(j_k, t) \) denote the data center accommodating job \( k \) in time interval \( t \), and notation \( q_k^i (i, i') \) denote the cost of moving job \( j_k \) from data center \( i \) to \( i' \). As the result of uncertainty of performance degradation and network consumption during job \( j_k \) migration, only core relevant factors are utilized for estimation [19]. The total job migration cost at time interval \( t \) is therefore expressed as

\[
C_{\text{migrate}}^{t} = \sum_{j_k \in J} q_k^i (d(j_k, t), d(j_k, t - 1))
= \sum_{j_k \in J} \text{res}_{j_k} \times \text{dRatio}_{j_k} \times \text{sen}_{j_k}.
\tag{4}
\]

where \( \text{res}_{j_k} \) represents memory size of job \( j_k \) migration, \( \text{dRatio}_{j_k} \) represents page dirty ratio of job \( j_k \) migration, and \( \text{sen}_{j_k} \) represents application sensitivity against job \( j_k \) migration.

Finally, the total cost over the whole time intervals on energy consumption and job migration can be calculated by:

\[
C = \sum_{t=1}^{T} \left( C_{i}^{\text{energy}} + C_{\text{migrate}}^{t} \right). \tag{5}
\]

Our proposal is designed for minimizing the total cost by making a job scheduling decision, i.e., \( \{J_1, J_2, \ldots, J_m\} \), at the beginning of each time interval, without the knowledge of future renewable energy generation \( e_{i,t} \) and job migration cost function \( q_k^i (i, i') \). The systems and variables in this paper are summarized in Table I.

### 3 REINFORCEMENT LEARNING BASED LOAD BALANCING

In this section, an RL-based method is proposed for dealing with the cost minimization issue. The method overview is first presented, followed by design details.

#### 3.1 Overview

RL is an approach to approximate the optimal reward by iteratively learning the feedback from historical decisions named actions. The learning process of RL consists of a sequence of actions and the corresponding rewards [14]. In each iteration, RL evaluates the expected effect of taking different actions by a value function. Then RL selects an action to execute, and observes the state that appears thereafter, the reward associated with this state that can be used to refine its value function. Generally, the reward is a metric of the benefit associated with an action in a certain state. Since RL do not need any priori knowledge, it is quite appropriate to the cost reduction among complex data centers.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D )</td>
<td>( {d_1, d_2, \ldots, d_n} ), a set of data centers</td>
</tr>
<tr>
<td>( e_{i,t} )</td>
<td>the energy cost at data center</td>
</tr>
<tr>
<td>( J )</td>
<td>( {j_1, j_2, \ldots, j_n} ), a set of big data jobs</td>
</tr>
<tr>
<td>( c_k )</td>
<td>the data arriving rate of job ( j_k )</td>
</tr>
<tr>
<td>( u_{i,t} )</td>
<td>the energy consumption of data center ( d_i ) at time interval ( t )</td>
</tr>
<tr>
<td>( r_{i,t}^{\text{cpu}} )</td>
<td>the utilized CPU resource of data center ( d_i ) at time interval ( t )</td>
</tr>
<tr>
<td>( p_{i,t}^{\text{brown}} )</td>
<td>the power of ( d_i ) when the utilization of it’s CPU is at 0%</td>
</tr>
<tr>
<td>( p_{i,t}^{\text{green}} )</td>
<td>the power of ( d_i ) when the utilization of it’s CPU is at 100%</td>
</tr>
<tr>
<td>( \text{res}_{j_k} )</td>
<td>memory size of job ( j_k ) migration</td>
</tr>
<tr>
<td>( \text{dRatio}_{j_k} )</td>
<td>page dirty ratio of job ( j_k ) migration</td>
</tr>
<tr>
<td>( \text{sen}_{j_k} )</td>
<td>application sensitivity against job ( j_k ) migration</td>
</tr>
<tr>
<td>( C_{\text{migrate}}^{t} )</td>
<td>the total job migration cost at time interval ( t )</td>
</tr>
<tr>
<td>( C )</td>
<td>the total cost</td>
</tr>
<tr>
<td>( s_t )</td>
<td>the system state in RL</td>
</tr>
<tr>
<td>( A )</td>
<td>a set of action ( a_t = {J_1, J_2, \ldots, J_m} ) in RL</td>
</tr>
<tr>
<td>( r_{t+1} )</td>
<td>the reward of action ( a_t ) in state ( s_t ) in RL</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>the discount factor of cumulative reward in RL</td>
</tr>
<tr>
<td>( Q^t(s_t, a_t) )</td>
<td>the action-value function in RL. It calculates the action-value of ( a_t ) in state ( s_t )</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>the pool size in SPS or RPS</td>
</tr>
<tr>
<td>( \tau )</td>
<td>the random discarding ratio in RPS</td>
</tr>
<tr>
<td>( \eta )</td>
<td>( \eta \in (0, 1) ) and controls the action selection in RL</td>
</tr>
<tr>
<td>( M )</td>
<td>the training epoches of RL</td>
</tr>
<tr>
<td>( F(s_t, a_t) )</td>
<td>the function trained by NN to approximate ( Q^\pi(s_t, a_t) )</td>
</tr>
<tr>
<td>( a_t^* )</td>
<td>the action that maximizes the output of function ( F(s_t, a_t) )</td>
</tr>
</tbody>
</table>

Our proposed algorithm is mainly designed to migrate jobs among geo-distributed data centers according to the feedback of historical job scheduling decisions. We abstract our algorithm as a job scheduler, whose interaction with job execution on data centers is shown in Fig. 2(a). First, the job scheduler selects an action and sends it to data centers. Then, data centers execute the action. Finally, data centers give feedback including state and reward to job scheduler for its learning. The temporal relations among these procedures is shown in Fig. 2(b). In the beginning phrase of every time interval \( t \), the job scheduler determines
TABLE 2:
THE INFORMATION EVERY STATE CONTAINS.

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s^t_1$</td>
<td>The CPU usage in current state.</td>
</tr>
<tr>
<td>$s^t_2$</td>
<td>The free RAM size in current state.</td>
</tr>
<tr>
<td>$s^t_3$</td>
<td>The free I/O bandwidth in current state.</td>
</tr>
<tr>
<td>$s^t_4$</td>
<td>The current weather.</td>
</tr>
<tr>
<td>$s^t_5$</td>
<td>The current electricity price.</td>
</tr>
</tbody>
</table>

Fig. 2: The scheme and procedures of load balancing.

Fig. 3: Training process.

3.2 Algorithm design

Given $s_t$ and $a_t$, we define an action-value function

$$Q^π(s_t, a_t) = \mathbb{E}\left[ \frac{1}{r_{t+1} + \lambda r_{t+2} + \lambda^2 r_{t+3} + \ldots} | s_t, a_t \right]$$

(7)

where $λ$ is a discount factor. Let $λ \in [0, 1]$ so that the rewards in the nearer future have larger weights. In each time interval, we need to select the action $a^*_t$ leading to the maximum value of function $Q^π(s_t, a_t)$, i.e.,

$$a^*_t = \arg \max_{a_t \in A} Q^π(s_t, a_t),$$

(8)

where $A$ is a set of action $a_t = \{J^t_1, J^t_2, \ldots, J^t_m\}$ that can be chosen in $s_t$.

Since future rewards are affected by many factors, such as resource utilization of each data center and local weather conditions as shown in Table II, the traditional reinforcement learning based on temporal differences [15] cannot accurately calculate $Q^π(s_t, a_t)$. Neural Network (NN) is deployed to train a function $F(s_t, a_t)$ that approximates the action-value function with high accuracy. NN can be considered as a composite function that regards state $s_t$ as input and outputs an action $a_t$. As shown in Fig. 3, an example neural network has 4 layers, and each layer contains a number of computing items called neurons. Each neurons receives values from all neurons in its upper layer, which are denoted by a vector $x$ and conducts a calculation in the form of $wx + b$, where $w$ and $b$ are called weight and bias.

The pseudo code of our proposed RL-based algorithm is shown in Algorithm 1. We start with an initial NN $F(s_t, a_t)$ with random weight $ω$ and bias $b$. In the beginning of each time interval, we generate a random number $z \in [0, 1]$ obeys uniform distribution. If $z < η$, we select the action $a^*_t$ that maximizes the output of function $F(s_t, a_t)$. Otherwise, we use the randomly selected action. After applying the action $a^*_t$, we can observe a system state $s_{t+1}$ at the end of time interval $t$, its associated reward $r_{t+1}$ will be used to retrain the NN based on the stochastic gradient descent (SGD) method [20].

As shown in Fig. 3, every circle corresponds to an example neural network. In the end phrase of time interval $t$, we generate a random number $z \in [0, 1]$ obeys uniform distribution. If $z < η$, we select the action $a^*_t$ that maximizes the output of function $F(s_t, a_t)$. Otherwise, we use the randomly selected action. After applying the action $a^*_t$, we can observe a system state $s_{t+1}$ at the end of time interval $t$, its associated reward $r_{t+1}$ will be used to retrain the NN based on the stochastic gradient descent (SGD) method [20].

As shown in Fig. 3, every circle corresponds to an example neural network. In the end phrase of time interval $t$, we generate a random number $z \in [0, 1]$ obeys uniform distribution. If $z < η$, we select the action $a^*_t$ that maximizes the output of function $F(s_t, a_t)$. Otherwise, we use the randomly selected action. After applying the action $a^*_t$, we can observe a system state $s_{t+1}$ at the end of time interval $t$, its associated reward $r_{t+1}$ will be used to retrain the NN based on the stochastic gradient descent (SGD) method [20].

As shown in Fig. 3, every circle corresponds to an example neural network. In the end phrase of time interval $t$, we generate a random number $z \in [0, 1]$ obeys uniform distribution. If $z < η$, we select the action $a^*_t$ that maximizes the output of function $F(s_t, a_t)$. Otherwise, we use the randomly selected action. After applying the action $a^*_t$, we can observe a system state $s_{t+1}$ at the end of time interval $t$, its associated reward $r_{t+1}$ will be used to retrain the NN based on the stochastic gradient descent (SGD) method [20].

As shown in Fig. 3, every circle corresponds to an example neural network. In the end phrase of time interval $t$, we generate a random number $z \in [0, 1]$ obeys uniform distribution. If $z < η$, we select the action $a^*_t$ that maximizes the output of function $F(s_t, a_t)$. Otherwise, we use the randomly selected action. After applying the action $a^*_t$, we can observe a system state $s_{t+1}$ at the end of time interval $t$, its associated reward $r_{t+1}$ will be used to retrain the NN based on the stochastic gradient descent (SGD) method [20].
Algorithm 1: Our proposed RL-based Algorithm

Input: $s_0, A = \{a_0, a_1, a_2, \cdots, a_T\}, \lambda, \eta, M$
Output: $(w, b)$

1. begin
2. initialize the NN $F(s_t, a_t)$ with random weight $\omega$ and bias $b$, $s_t = s_0$
3. for $i \in (1, 2, \cdots, T)$ do
4. for $j \in (1, 2, \cdots, M)$ do
5. generate random number $z \in [0, 1]$
6. if $z < \eta$ then
7. select action $a_t^* \in A$ that maximizes the output of function $F(s_t, a_t)$
8. else
9. randomly select action $a_t^* \in A$
end
10. execute $a_t^*$, and observe reward $r_{t+1}$ and state $s_{t+1}$
11. retrain $(s_t, a_t^*, r_{t+1})$
12. begin
13. retrain NN using $(s_t, a_t^*, r_{t+1})$ by SGD
14. end
15. set $s_t = s_{t+1}$
end
19. end

4 PERFORMANCE ENHANCEMENT

4.1 Random pool sampling

In Algorithm 1, we retrain the NN by using the stochastic gradient descent method that requires stochastic sampling (SS) [20] of training data. However, such requirement cannot be met because only one data record is generated and used in retraining in the end phrase of every time interval. To overcome this weakness, Simple Pool Sampling (SPS) [21] is proposed to maintaining a pool to accumulate the training data. When the pool is full, it performs training process and empties the pool. The weakness of SPS is that only the data recently come can be sampled. Alternatively, we propose a random pool sampling (RPS) scheme to approximate SS such that the accuracy of NN can be further increased. Since the RPS maintains a pool to store data, we also replace SGD with mini-batch stochastic gradient descent which generally has better accuracy than SGD [22]. As shown in Fig. 4, we maintain a pool of historical data and use them to retrain the NN based on mini-batch stochastic gradient descent. When the pool is full, we randomly discard some data records in the pool. Note that the sampling method used in Algorithm 1 is a particular case of RPS via installing the pool size to 1.

Combining RPS with mini-batch SGD, we propose a novel algorithm called retrainOnBatch to replace the function retrain in Algorithm 1. The pseudo codes are shown in Algorithm 2.

In Algorithm 2, we first check whether the sampling pool is full or not. If it is full, we shuffle all data to generate a mini-batch that will be used by SGD to retrain the NN, as shown in lines 2 and 3. Then, we randomly discard $\sigma \times \tau$ data records, where $\sigma$ is the pool size and $\tau$ is the discarding ratio. If it is not full, we just insert the new data records into the pool.

To theoretically evaluate the performance of different data sampling methods, we use the metric of Bhattacharyya distance [23] that is defined as follows.

Definition 1. Bhattacharyya Distance is a measurement of the similarity between two discrete probability distributions. For two distributions $p^*$ and $q^*$ in the same region $X$, the Bhattacharyya distance can be written as

$$
D_B(p^*, q^*) = -\ln(BC(p^*, q^*)),
$$

where $BC$ is called Bhattacharyya coefficient that can be calculated by:

$$
BC(p^*, q^*) = \sum_{x \in X} \sqrt{p^*(x)q^*(x)}.
$$

The distributions of sampling data under SS, SPS, and RPS are denoted by $p_{SS}, p_{SPS}$ and $p_{RPS}$, respectively. We show the superiority of RPS to SPS, which brings further increase in accuracy, in the following theorem.

Theorem 1. $D_B(p_{RPS}, p_{SS}) \leq D_B(p_{SPS}, p_{SS})$

Proof. The pool is assumed full at time interval $t$ with no generality loss, and mini-batch based SGD samples $\sigma$ shuffled tuples from the pool. Since tuples are generated one by one, we denote tuples $(\text{state}_t, \text{reward}_t, \text{action}_t)_{t \in [0, T]}$ with sequence as

$$
(u_0, u_1, \cdots, u_T).
$$

Then, we denote a sequence $P$ as

$$
(p_0^*, p_1^*, \cdots, p_T^*),
$$

where $p_i$ is the probability to be sampled of the tuple $u_i$. The $P_{SS}$ can be denoted as

$$
P_{SS} = (\frac{\sigma}{T}, \frac{\sigma}{T}, \cdots, \frac{\sigma}{T}).
$$
According to the definition of SPS and RPS, $P_{SPS}$ is formulated as

$$P_{SPS} = (0, 0, \cdots, 1, 1, \cdots, 1, 0, \cdots, 0),$$

and $P_{RPS}$ is formulated as

$$P_{RPS} = ((1-\tau)^\kappa, \cdots, (1-\tau)^\kappa, \cdots, (1-\tau)^{\kappa-1}, \cdots, (1-\tau)^{\kappa-1}, \cdots, (1-\tau)^{2}, \cdots, (1-\tau)^{2}, 1, \cdots, 1, 0, \cdots, 0),$$

where $\kappa$ represents the times of when the pool is full, $\kappa \propto t$.

Use Eq. (10) to calculate the Bhattacharyya Coefficient between SPS and SS, we can get

$$BC(p_{SPS}, p_{SS}) = \sigma \frac{\sqrt{\sigma}}{T},$$

Similarly, the coefficient between RPS and SS is calculated by

$$BC(p_{RPS}, p_{SS}) = \tau \sigma \frac{\sqrt{\sigma}}{T} + \tau \sigma \frac{\sqrt{\sigma}}{T} \sqrt{(1-\tau)^1} + \cdots$$

$$+ \tau \sigma \frac{\sqrt{\sigma}}{T} \sqrt{(1-\tau)^{\kappa-1}}$$

$$+ \sigma \frac{\sqrt{\sigma}}{T} \sqrt{(1-\tau)^{\kappa}}$$

$$= \sigma \frac{\sqrt{\sigma}}{T} \tau (1 + \sqrt{1-\tau} + \sqrt{(1-\tau)^2}$$

$$+ \cdots + \sqrt{(1-\tau)^{\kappa-1}} + \sqrt{(1-\tau)^{\kappa}}.$$  \hspace{1cm} (17)

When the $t$ is sufficiently large, we can get

$$BC(p_{RPS}, p_{SS}) = \sigma \frac{\sqrt{\sigma}}{T} (1 + \sqrt{1-\tau})$$

$$> \sigma \frac{\sqrt{\sigma}}{T} = BC(p_{SPS}, p_{SS}).$$  \hspace{1cm} (18)

Combine Eq. (18) with Eq. (9), we can get

$$D_B(p_{RPS}, p_{SS}) \leq D_B(p_{SPS}, p_{SS}),$$

which finishes the proof.

### 4.2 Unidirectional bridge network structure

In this subsection, we further enhance our proposed RL-based algorithm by proposing a unidirectional bridge network (UBN) structure to accelerate the training process of NNs. It is motivated by the fact that neurons in the NN contain useful knowledge that can be reused in training the new NN.

In the proposed UBN structure, shown in Fig. 5, we create transfer layers between the new NN and the one trained previously. Every transfer layer contains a number of neurons that randomly collect the outputs from the trained NN. The outputs of transfer layers are sent to the new NN. Every hidden layer and output layer in the new NN receives data from both the previous layer and a transfer layer.

![Fig. 5: Unidirectional bridge network structure.](image)

Since the transfer layers bring extra neurons and associated connections, we propose the random dropout [24] strategy to reduce the number of retrained neurons. As shown in Fig. 6, the dropout strategy is to randomly remove some neurons and their connections. This transfer layer can be considered as a layer with two neurons in training. Note that the dropout strategy runs at the beginning of every iteration of training, and the dropped neurons just do not participate in the current training iteration. In every iteration of training process, the number of dropped neurons is kept at a stable value, but the neurons are uncertain.

### 5 Experiments

To study the benefits of our proposed RL-based algorithm, extensive experiments based on real world data sets are performed in this section.

#### 5.1 Experimental setup

In our experiments, we use the following real-world data sets including data center location, workload, renewable energy generation, and electricity price of traditional power grid.

1) Workload: We use the Google cluster-usage traces [25] collected on May, 2011, which contain 29-day job information on a cluster of about 12.5k machines.

2) Renewable energy generation: We consider three data centers located in Preovit in New Mexico (NM), Phoenix in Arizona (AZ) and Los Angeles in California (CA), where renewable energy is generated according to weather conditions published by National Renewable Energy Laboratory [26]. We use the energy cost and renewables for the same
time. The renewable energy data includes the all-day generation of solar, water, and wind.

3) Electricity price: The electricity price is obtained from the website of Energy Information Administration [3].

To study the influence of NN on the proposed RL-based algorithm, the discount factor $\lambda$ of cumulative reward in RL is set as 0.1, 0.3, 0.5, 0.7, 0.9, and the $\eta$, which controls the action selection in RL, is set as 0.1, 0.3, 0.5, 0.7, 0.9. We consider four NN structures with different combinations of type, number of layers, and dropout policies in our experiments. As shown in Table III, one structure uses Convolutional Neural Network (CNN) [27] whose kernel size is set to 3, and others adopt multilayer perception (MLP) [24] with one kernel. The three structures using MLP have different number of layers and dropout policies. For comparison, we also show the performance of following algorithms as benchmarks.

- **Round robin (RR):** It schedules jobs to data centers in a round-robin manner without any information about renewable energy generation [28].

- **MinBrown (MB):** The MB algorithm is developed for the minimizing brown energy consumption based on workload scheduling strategy [29].

- **Temporal Difference (TD):** TD is the default algorithm used by traditional RL [15].

### 5.2 Experimental results

We analyze the experimental results from the aspect of the affects of network structures, the performance of RPS, the impacts of UBN, the comparison with other algorithms, and the impact of $\lambda$ and $\eta$. Moreover, the convergence of the reinforcement learning is discussed.

1) **The affects of network structures**

We first study the total cost and convergence speed of our proposed algorithm with different NN structures. In Fig. 7(a), we normalize all cost to the result of structure D. The normalized cost of structures A, B, and C is lower than 1, indicating that they outperform D. Structure B has the lowest cost because of the advantages of MLP with larger number of layers. The convergence is shown in Fig. 7(b), where we observe that structure D has the fasted converge speed, but leading to a highest cost and lowest average action-value. Meanwhile, structure B converges slower, but it has lowest cost and highest average action-value. Therefore, the structure B can make a good tradeoff between cost, convergence speed, and convergence effect, which will be adopted in the following experiments.

2) **The performance of RPS**

We then investigate the performance of RPS by changing the discarding ratio. As shown in Fig. 8(a), the lowest cost can be achieved by setting the discarding ratio to 50%. That is because under larger discarding ratio, many data records in the pool will be dropped when it is full, leading to an obvious temporal relation between two mini-batch, i.e., the data in latter mini-batch are always later than that in the previous. On the other hand, retraining based on mini-batch SGD will be frequently triggered under smaller discarding ratio, and a large amount of redundant data will be generated. Furthermore, the results in Fig. 8(b) demonstrate that setting discarding ratio to 50% can guarantee the fastest convergence speed and the highest average action-value.

Therefore, when the discarding ratio is set to 50%, the RPS converges the best.

3) **The impacts of UBN**

We investigate the acceleration of UBN under different dropout policies. As shown in Fig. 9, the algorithm without UBN slowly converges, but others with dropout has higher average action-value and faster convergence speed, since dropout makes the NN more adaptive by preventing the solution from sinking into a local optimum. UBN with 50% dropout has highest average action-value and converges the best.

4) **The comparison with other algorithms**

In this set of experiments, we choose structure B with randomly discarding ratio at 50%, and a pre-trained net-

---

**TABLE 3: DIFFERENT NETWORK STRUCTURES**

<table>
<thead>
<tr>
<th>Structure</th>
<th>Type</th>
<th>no. of layers</th>
<th>dropout</th>
<th>kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>CNN</td>
<td>13</td>
<td>Yes</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>MLP</td>
<td>13</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>9</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>5</td>
<td>No</td>
<td>1</td>
</tr>
</tbody>
</table>
work is connected to the network to be trained using transfer layers with 50% dropout. Comparison among our proposed RL-based Algorithm (Strategy Learning Algorithm, SLA), advanced strategy learning algorithm (ASLA) that combines SLA with algorithm 2, and benchmark algorithms are shown in Fig. 10. As shown in Fig. 10, ASLA has better performance on the normalized average cost saving than most of benchmark algorithms and SLA. In ASLA, the job scheduler has the exploration mechanism controlled by $\eta$, which prevent the data centers from an extreme load imbalance. Specially, the average action-value comparison among SLA, ASLA, and TD is shown in Fig. 10(b). According to Fig. 10(b), SLA and ASLA both perform better than TD that uses tabular action-value function.

6 RELATED WORK

6.1 Green data centers

The energy reduction issue of a single data center are undergoing extensively studied in recent years. For example, Wu et al. [4] have proposed to use dynamic CPU voltage to minimize power consumption of data centers. A similar approach could dynamically adjust the performance of data centers according to the load [5].

The other branch focuses on using task distribution technical across geo-distributed data centers to achieve cost minimization. Rahman et al. [30] presented a survey of geographic load balancing in smart grid. Liu et al. [31] proposed GreenCloud, integrating online monitoring and VM placement optimization. Zhang et al. [32] developed Hierarchical EneRgy Optimization (HERO) scheme for the

5) The impact of $\lambda$ and $\eta$ of ASLA

The normalized costs of ASLA with different $\lambda$ and $\eta$ are shown in Fig. 11. According to Fig. 11(a), the costs are basically not affected by the value of $\lambda$. We can see that the normalized cost is influenced by $\eta$ obviously in Fig. 11(b). Since the low $\eta$ controls ASLA to concern more load rather than cost, when the $\eta$ is set to 0.9, ASLA saves the most cost.
However, there are only a few researches of combining reinforcement learning with geographic data centers load balancing. Lin et al. [15] proposed a reinforcement learning-based framework to optimize the power management, but they just uses a simple value function, which may not be able to fit the data centers appropriately.

6.3 Load balancing

Load balancing is a research hotspot in geo-distributed data centers. Tang et al. [40] proposed a heuristic scheduling approach to balance the workload dynamically. Chen et al. Wang et al. [41] proposed a method of load balancing towards multimedia big data in data centers. The method is based on hybrid stream and CNN. [42] proposed a network topology aware based approach to perform load balancing, which takes advantage of parallelization and is able to speed up load balancing process. Gupta et al. [43] applied Ant Colony Optimization in load balancing. The algorithm performs load balancing by detecting overloaded and underloaded servers. Shao et al. [44] investigated the optimal load balancing problem and took transmission delay into account.

Different from these previous work, our approach uses reinforcement learning to distribute jobs coming to the data centers, and replaces the value function with NN. Different from these studies, our proposal are self-learn online algorithms. In addition, we design a random pool sampling approach to dismiss the time correlation among rewards and speed up the convergence of the NN. We also propose a new network structure for receive the priori knowledge from other network, which brings significant efficiency improvement. We propose the combination of RL and NN for the first time.

7 CONCLUSION AND FUTURE WORK

In this paper, we have investigated a big data scheduling problem for reducing the cost of geo-distributed data centers. An RL based job scheduling algorithm is proposed with NN, and two techniques are developed to enhance the performance of our proposal. Specifically, we propose RPS to retrain the NN via accumulated training data, and design a novel UBN structure for further enhancing the training speed. Extensive experiments show that our proposal is able to reduce the data centers’ cost significantly compared with some benchmark algorithms.

Motivated by this paper, there are many interesting directions that can be studied. With respect to the design of parallel algorithms, one aspect is to use multiple processes to accelerate the learning speed of job scheduler. Another aspect is to investigate the combination of ensemble learning and reinforcement learning for rapid deployment. In the field of decentralized distributed systems, multi-agent reinforcement learning is an aspect can be studied. As for the exploration policy, our approach uses simple greedy algorithm based on probability. However, exploration policy is also important in practice. Further studies will focus on reducing the cost of cloud service.
ACKNOWLEDGMENT
This work is supported by National China 973 Project (2015CB352401); NSFC (61572262); China Postdoctoral Science Foundation (2017M610252); China Postdoctoral Science Special Foundation (2017T100297); JSPS KAKENHI (16K16038); and Strategic Information and Communications R&D Promotion Programme (SCOPE No.162302008), MIC, Japan.

REFERENCES


Chenhuan Xu is a postgraduate student in the School of Internet of Things, Nanjing University of Posts and Telecommunications, China. His current research interests include big data, cloud computing, and machine learning.

Kun Wang [M’13-SM’17] received the B.Eng. and Ph.D. degree in computer science from Nanjing University of Posts and Telecommunications, Nanjing, China, in 2004 and 2009, respectively. From 2013 to 2015, he was a Postdoc Fellow in Electrical Engineering Department, University of California, Los Angeles (UCLA), CA, USA. In 2016, he was a Research Fellow in the School of Computer Science and Engineering, the University of Aizu, Aizu-Wakamatsu City, Fukushima, Japan. He is currently a Research Fellow in the Department of Computing, the Hong Kong Polytechnic University, Hong Kong, China, and also a Full Professor in the School of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing, China. He has published over 100 papers in refereed international conferences and journals. He has received Best Paper Award at IEEE GLOBECOM’16. He serves as Associate Editor of IEEE Access, Editor of Journal of Network and Computer Applications, EAI Transactions on Industrial Networks and Intelligent Systems and Guest Editors of IEEE Access, Future Generation Computer Systems, Peer-to-Peer Networking and Applications, and Journal of Internet Technology. He was the symposium chair/co-chair of IEEE IECION16, IEEE EEEIC16, IEEE WCSP16, IEEE CNCC17, etc. His current research interests are mainly in the area of big data, wireless communications and networking, smart grid, energy Internet, and information security technologies. He is a Senior Member of IEEE and Member of ACM.

Minyi Guo [F’17] received the Ph.D. degree in computer science from the University of Tsukuba, Tsukuba, Japan. He is currently a Chiyuan Chair Professor in Shanghai Jiao Tong University, Shanghai, China. His research interests include pervasive computing, parallel and distributed processing, and parallelizing compilers. In 2007, he received the Recruitment Program of Global Experts and Distinguished Young Scholars Award from the National Natural Science Foundation of China. He is on the editorial board of IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Computers. He is a fellow of IEEE.

Song Guo [M’02-SM’11] is a Full Professor at Department of Computing, The Hong Kong Polytechnic University. He received his Ph.D. in computer science from University of Ottawa and was a professor with the University of Aizu. His research interests are mainly in the areas of big data, cloud computing and networking, and distributed systems with over 400 papers published in major conferences and journals. His work was recognized by the 2016 Annual Best of Computing: Notable Books and Articles in Computing in ACM Computing Reviews. He is the recipient of the 2017 IEEE Systems Journal Annual Best Paper Award and other five Best Paper Awards from IEEE/ACM conferences. Prof. Guo was an Associate Editor of IEEE Transactions on Parallel and Distributed Systems and an IEEE ComSoc Distinguished Lecturer. He is now on the editorial board of IEEE Transactions on Emerging Topics in Computing, IEEE Transactions on Sustainable Computing, IEEE Transactions on Green Communications and Networking, and IEEE Communications. Prof. Guo also served as a General, TPC and Symposium Chair for numerous IEEE conferences. He currently serves as an officer for several IEEE ComSoc Technical Committees and a director in the ComSoc Board of Governors.

Rui Xia received her BS degree from Shanghai Jiao Tong University, Shanghai, China, and now is working as an automation developer in Autodesk(China)Software Research and Development Co., Ltd., Shanghai, China. Her research interests mainly focus on machine learning, big data and web service.

Peng Li [M’12] received his BS degree from Huazhong University of Science and Technology, China, in 2007, the MS and PhD degrees from the University of Aizu, Japan, in 2009 and 2012, respectively. He is currently an Associate Professor in the University of Aizu, Japan. His research interests mainly focus on wireless communication and networking, specifically wireless sensor networks, green and energy-efficient mobile networks, and cross-layer optimization for wireless networks. He also has interests on cloud computing, big data processing and smart grid. He is a member of IEEE.