Resource-aware distributed differential evolution for training expensive neural-network-based controller in power electronic circuit

Resource-Aware Distributed Differential Evolution for Training Expensive Neural-Network-Based Controller in Power Electronic Circuit

Xiao-Fang Liu, *Member, IEEE*, Zhi-Hui Zhan, *Senior Member, IEEE*, and Jun Zhang, *Fellow, IEEE*

***Abstract*— The neural-network (NN)-based control method is a new emerging promising technique for controller design in a power electronic circuit (PEC). However, the optimization of NN-based controllers (NNCs) has significant challenges in two aspects. The first challenge is that the search space of the NNC optimization problem is such complex that the global optimization ability of the existing algorithms still needs to be improved. The second challenge is that the training process of the NNC parameters is very computationally expensive and requires a long execution time. Thus, in this article, we develop a powerful evolutionary computation-based algorithm to find a high-quality solution and reduce computational time. First, the differential evolution (DE) algorithm is adopted because it is a powerful global optimizer in solving a complex optimization problem. This can help to overcome the premature convergence in local optima to train the NNC parameters well. Second, to reduce the computational time, the DE is extended to distribute DE (DDE) by dispatching all the individuals to different distributed computing resources for parallel computing. Moreover, a resource-aware strategy (RAS) is designed to further efficiently utilize the resources by adaptively dispatching individuals to resources according to the real-time performance of the resources, which can simultaneously concern the computing ability and load state of each resource. Experimental results show that, compared with some other typical evolutionary algorithms, the proposed algorithm can get significantly better solutions within a shorter computational time.**

***Index Terms*— Distributed differential evolution (DDE), neural-network-based controller (NNC), power electronic circuit (PEC), resource-aware strategy (RAS).**

I. Introduction

THE design of power electronic circuits (PECs) is a fundamental and significant research topic in power electronics [1]. However, the controller design in PEC is a challenging issue due to the time-varying and piecewise linear characteristics of the power conversion stage (PCS) [2]. Currently, there are two common methods to design the PEC controller. One is the traditional feedback-network (FN)-based control method [3], and the other is the new emerging modern machine learning-based control method [4]. The FN-based controllers (FNCs) use a set of electronic components (e.g., resistors, inductors, and capacitors) to generate a control signal to feedback to the PCS. Therefore, this is a hardware control method. Instead, the machine learning-based control method uses a software method (e.g., an approximation algorithm) to simulate the control signal. Particularly, as neural networks (NNs) have become a very popular machine learning technique for control tasks due to their strong ability of function approximation [5]–[11], the NN-based controllers (NNCs) have been used in PEC. The NNCs use a well-trained NN to generate control signals to the PCS. Compared with the FNCs, the design flow of the NNCs is intuitional [12], [13]. NNCs not only avoid the requirement of deep understanding of the PCS and time-consuming derivation of control laws for complex PEC systems but also present better performance. NNCs have gained increasing attention from academic and industrial communities and have become a new trend in PEC design [14]. However, there are still challenges to the NNC design.

First, the performance of the controllers is greatly sensitive to the NN parameters. Backpropagation approaches are popular for NN training [15]–[18]. However, backpropagation approaches require a large training data set and are easily trapped in local optima [19]. Particularly, the training data set is also difficult to construct for PEC since the excepted control signal (output) for a given input (state variables from the PCS) is hard to obtain. Alternatively, evolutionary computation (EC) has also been applied to NN optimization, termed evolutionary NN (ENN) [20]–[22], such as cost evaluation [23] and parameter optimization [24]. Compared with backpropagation

Manuscript received 17 March 2020; revised 7 December 2020; accepted 15 April 2021. Date of publication 7 May 2021; date of current version 28 October 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2102102, in part by the Outstanding Youth Science Foundation under Grant 61822602, in part by the National Natural Science Foundations of China (NSFC) under Grant 61772207 and Grant 61873097, in part by the Key-Area Research and Development of Guangdong Province under Grant 2020B010166002, and in part by the Guangdong Natural Science Foundation Research Team under Grant 2018B030312003. *(Corresponding authors: Zhi-Hui Zhan; Jun Zhang.)*

Xiao-Fang Liu is with the Institute of Robotics and Automatic Information System, College of Artificial Intelligence, Nankai University, Tianjin 300350, China, and also with the Tianjin Key Laboratory of Intelligent Robotics, Nankai University, Tianjin 300350, China.

Zhi-Hui Zhan is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, also with the Pazhou Laboratory, Guangzhou 510330, China, and also with the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, South China University of Technology, Guangzhou 510006, China (e-mail: zhanapollo@163.com).

Jun Zhang is with Zhejiang Normal University, Jinhua 321004, China, and also with the Chaoyang University of Technology, Taichung 41349, Taiwan.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2021.3075205.

Digital Object Identifier 10.1109/TNNLS.2021.3075205

methods, EC is gradient-free and has a stronger global searchability. However, since the search space of NN optimization is large and complex, an efficient global optimization method is in great need. Second, the computational time of controller optimization is very long since the function evaluation (FE) of NNC optimization is expensive. Therefore, how to reduce computational time is also a challenging issue.

In order to relieve the above two challenges, we focus on developing an optimization algorithm with stronger global searchability and shorter computational time for PEC controller design. In terms of global optimization, a differential evolution (DE) algorithm is adopted to find the approximately optimal parameters of the controllers. DE is a branch of EC algorithms, which was proposed by Storn and Price [25] and has been fast developed to become an attractive global optimizer in recent years due to its simple algorithm structure and easy implementation [26]–[30]. Since the progress of the exploration of DE to PEC design is at a slow pace, it is greatly interesting and of practical value to extend DE to PEC optimization. The adopted DE algorithm provides a simple yet efficient approach to solve PEC, attempting to help overcome the premature convergence of other optimization methods. In addition, using DE to optimize the NNC in PEC also provides a kind of powerful approach for ENN.

To further reduce the computational time, it is promising to leverage the inherited parallelism advantage of DE for extending traditional centralized DE to distributed DE (DDE) [31]. This way, different individuals can be dispatched on different computing resources in the evaluation step to calculate fitness values concurrently to reduce computational time. Although this is an intuitive idea, the difficulty is how to efficiently assign the individuals to the resources to reduce the computational time as much as possible. A very straightforward and simple way is to evenly divide the individuals into groups and dispatch each group to one resource [32]–[34]. However, it may be inefficient since it ignores the computing abilities and load states of the different distributed resources. Some load-balance-based methods have also been proposed, such as CPU-utilization-based distribution mode [35] and dynamic distribution strategy for heterogeneous tasks on homogeneous resources [36]. However, they neglect the computing abilities of the resources. More efficiently, in this article, we develop a resource-aware strategy (RAS) to adaptively dispatch individuals to resources according to the real-time performance of the resources, simultaneously concerning both the computing ability and load state of each resource. The proposed RAS is incorporated into DDE to form resource-aware DDE (RADDE).

The proposed RADDE is applied to solve two PEC instances for verifying its effectiveness. One instance with an NNC is used to show the advantage of RADDE on NNC optimization in terms of solution quality and computational time. Since NNCs require a very long time for optimization, it is impracticable to perform a large number of experimental comparisons. To overcome this, another instance with an FNC using shorter computational time is taken to evaluate the performance of RADDE and its acceleration in computational time under different resource environments. Experimental results show that the proposed RADDE algorithm has generally better performance than the compared algorithms. The main contribution of this article is to propose a DDE with a strong global search ability to find good parameters for controllers of PEC and develop a RAS to help the DDE significantly shorten the computational time.

The rest of this article is organized as follows. Section II introduces the DE algorithm and the PEC problems. Section III describes the proposed RADDE algorithm. Section IV verifies the performance and time acceleration of RADDE on a PEC with an FNC. In Section V, the RADDE is applied to a recent NNC. Finally, conclusions are summarized in Section VI.

II. Background

A. DE Kernel

DE uses a population to search for the global optimum through multiple iterations [41]. In the initialization, N individuals that represent solution vectors $X_i = [x_{i1}, x_{i2}, \ldots, x_{iD}]$ are randomly generated in the search space, where $1 \le i \le N$ is the individual index and D is the problem dimensionality. Each dimension x_{id} is within a certain search range $[x_{\min,d}$, $x_{\max,d}]$ determined by the problem. At every generation, each individual X_i performs a mutation operator to generate a new vector $V_i = [v_{i1}, v_{i2}, \ldots, v_{iD}]$ by using the difference information of two other individuals as

$$V_i = X_{r1} + F(X_{r2} - X_{r3}) \tag{1}$$

where F is the "*amplification factor*" parameter to control the scale of the differential vector and is commonly set within the range of [0, 1], while $r1$, $r2$, and $r3$ indicate different individual indexes and are also different from individual i. Note that, if any dimension v_{id} of V_i violates the search range, it will be set as the value of the corresponding violated boundary.

In the following, DE performs the crossover operation on the solution vectors X_i and V_i to obtain a new solution vector $U_i = [u_{i1}, u_{i2}, \ldots, u_{iD}]$ as:

$$u_{id} = \begin{cases} v_{id}, & \text{if rand(0,1)} \le \text{CR or } d = r_i \\ x_{id}, & \text{otherwise} \end{cases} \tag{2}$$

where CR is the "*crossover rate*" parameter and r_i is a randomly selected dimension subject to $r_i \in \{1, 2, \ldots, D\}$ to ensure that at least one dimension of V_i will enter the new solution vector U_i.

After the crossover operation, a selection operation is applied on the vectors X_i and U_i, and the better one will be allowed to enter the next generation, as in (3) for a minimization problem

$$X_i = \begin{cases} U_i, & \text{if } \text{fit}(U_i) \le \text{fit}(X_i) \\ X_i, & \text{otherwise.} \end{cases} \tag{3}$$

Specifically, for a maximization problem, X_i will be replaced by U_i if U_i has a greater or equal fitness value; otherwise, X_i enters the next generation.

All the individuals perform these three operations generation by generation until meeting the termination condition.

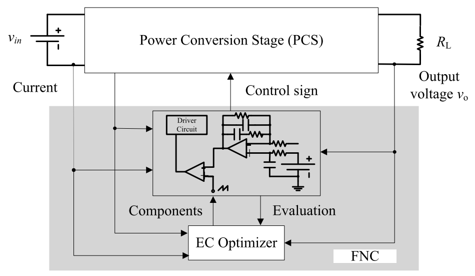

Fig. 1. Block diagram of a PEC with traditional FNC.

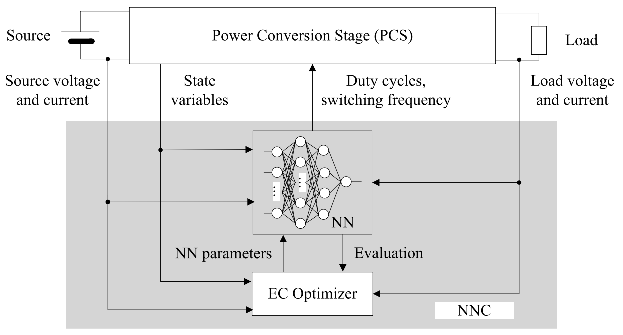

Fig. 2. System architecture of a PEC with NNC.

B. PEC With FNC

A PEC usually uses multiple circuit components to implement its function. The basic block diagram of a PEC with an FNC is shown in Fig. 1. There are two parts, i.e., PCS and FNC, in the circuit [37]. Particularly, the PCS part accepts power from the input source v_{in} and is responsible for transmitting it to the output load R_L, while the FNC coordinates the output voltage v_o to control the circuit performance. Usually, a control signal is given according to the received system state from the PCS, through the cooperation of multiple resistors, inductors, capacitors, and other components in the FNC. The component values in the FNC greatly influence the steady state of the circuit and should be well optimized.

C. PEC With NNC

Like traditional circuits, a PEC with an NNC also consists of two parts, named PCS and NNC, as shown in Fig. 2. The NNC accepts state variables from the PCS as input and outputs control signals to the PCS for circuit stableness. To well control the circuit, the NNC needs to be trained for obtaining optimal NN parameters. Given the circuit model and fitness function (calculation of training errors), the training of the NNC is actually a parameter optimization problem. The parameters of the NNC, i.e., the weights and biases of the NN, can be optimized by the EC engine. This article codes the parameters of the NNC as the variables of a solution and uses RADDE as the optimizer for training.

III. Proposed Algorithm

A. Representation Scheme and Objective Function

To optimize controllers, the electronic components in an FNC or the NN parameters in an NNC are represented as variables in solution vectors. Each individual X is coded as

$$X = [x_1, x_2, \ldots, x_D] \tag{4}$$

where x_d is the dth optimization variable and D represents the variable dimension. As mentioned in Section I, two different PEC instances are used for the test. The solution representation and fitness functions of the two PEC instances are presented in the following. More details of the two PEC instances will be described in Sections IV and V.

1) Solution Representation and Fitness Function of FNC: For a traditional FNC, each individual is encoded as

$$X = \left[\overline{R}\ \overline{I}\ \overline{C}\right] = [x_1, x_2, \ldots, x_D] \tag{5}$$

where $\overline{R} = [R_1, R_2, \ldots, R_{|R|}]$, $\overline{I} = [I_1, I_2, \ldots, I_{|I|}]$, and $\overline{C} = [C_1, C_2, \ldots, C_{|C|}]$ represent the component values of resistors, inductors, and capacitors, respectively, and $D = |R|+|I|+|C|$ is the problem dimension.

The optimization objective is mainly to reduce the settling time (ST) and control the overshoot [37] and can be formulated as

$$\begin{aligned}\max F(X) &= \sum_{R_L=R_{L\_\min},\delta R_L}^{R_{L\_\max}} \sum_{v_{\text{in}}=v_{\text{in}\_\min},\delta v_{\text{in}}}^{v_{\text{in}\_\max}} [f_1(R_L, v_{\text{in}}, X) + f_2(R_L, v_{\text{in}}, X) \\ &\quad + f_3(R_L, v_{\text{in}}, X)] + f_4(X)\end{aligned} \tag{6}$$

where $R_{L\_\min}$ and $R_{L\_\max}$, and $v_{\text{in}\_\min}$ and $v_{\text{in}\_\max}$ represent the minimal and maximal values of R_L and v_{in}, respectively, and δR_L and δv_{in} are user-defined step lengths in varying the values of R_L and v_{in}. Specifically, f_1 measures the steady-state error of the output voltage v_o and is defined as

$$f_1(R_L, v_{\text{in}}, X) = K_1 e^{-E_2/K_2} \tag{7}$$

$$E_2^{(r)} = \sum_{m=1}^{N_s} \left[v_o'^{(r)}(m) - v_{\text{ref}}\right]^2 \tag{8}$$

where v_o' is the value of v_o after the signal conditioning and is achieved by using the iterative Secant method [38] to perform a time-domain simulation that terminates if $E_2^{(r)}$ is less than a given tolerance ε and the system has reached the steady state; $E_2^{(r)}$ estimates the error between v_o' and v_{ref} by N_s simulated values in the rth iteration, K_1 is the maximum attainable value of f_1, and K_2 controls the sensitivity of f_1 to E_2; and f_2 measures the transient response of $v_d = v_{\text{ref}} - v_o'$, considering minimizing the maximum overshoot (OV), maximum undershoot (UV), and the ST, by

$$\begin{aligned}f_2(R_L, v_{\text{in}}, X) &= \text{OV}(R_L, v_{\text{in}}, X) \\ &\quad + \text{UV}(R_L, v_{\text{in}}, X) + \text{ST}(R_L, v_{\text{in}}, X)\end{aligned} \tag{9}$$

$$\text{OV}(R_L, v_{\text{in}}, X) = \frac{K_3}{1 + e^{(M_p - M_{p0})/K_4}} \tag{10}$$

$$\text{UV}(R_L, v_{\text{in}}, X) = \frac{K_5}{1 + e^{(M_v - M_{v0})/K_6}} \tag{11}$$

$$\text{ST}(R_L, v_{\text{in}}, X) = \frac{K_7}{1 + e^{(T_s - T_{s0})/K_8}} \tag{12}$$

$$v_d = v_{\text{ref}} - v_o' \tag{13}$$

where K_3, K_5, and K_7 are the maximum available values of OV, UV, and ST, respectively, K_4 and K_6 are passband constants, K_8 controls the sensitivity, M_{p0} is the maximum overshoot, M_p is the current overshoot, M_{v0} is the expected maximum undershoot, M_v is the current undershoot, T_{s0} is a constant, and T_s is the real ST required until $|v_d|$ falls within $0.01 \times \sigma$ band; f_3 controls the steady-state ripple voltage on the output v_o by

$$f_3(R_L, v_{\text{in}}, X) = K_9 e^{-A_1/K_{10}} \tag{14}$$

where K_9 is the maximum value of f_3, K_{10} is the decay constant, and A_1 represents the ripple voltage areas outside the tolerance band $\pm v_o$ around the expected value in the N_s simulated values; f_4 measures the dynamic behaviors during the large-signal change by

$$f_4(X) = \sum_{i=1}^{N_T} \text{OV}\big(R_{L,i}, v_{\text{in},i}, X\big) + \text{UV}\big(R_{L,i}, v_{\text{in},i}, X\big) + \text{ST}\big(R_{L,i}, v_{\text{in},i}, X\big) \tag{15}$$

where N_T is the number of input and load disturbances used to evaluate the fitness. Through maximizing the $F(X)$, we can obtain the optimal component values. Please refer to [37] for more details of the fitness function.

2) Solution Representation and Fitness Function of NNC: For an NNC, the representation of each individual is a vector of the NN parameters as

$$\begin{aligned} X &= \big[w_1, \ldots, w_{|\text{weights}|}, b_1, \ldots, b_{|\text{biases}|}\big] \\ &= [x_1, x_2, \ldots, x_D] \end{aligned} \tag{16}$$

where all w's are the weights of the connections in the whole NN, all b's are the biases in the neurons of the hidden and output layers, |weights| is the number of weights, |biases| is the number of biases, and $D = |\text{weights}| + |\text{biases}|$ is the problem dimension.

To find the optimal NN parameters, we need to evaluate the system performance for each solution of NN parameters. Given parameter values X for the NNC, a complete system is built. The system performance is evaluated using three indexes, i.e., output voltage, inductor current limit, and fluctuation of the duty cycle. Herein, $|S_{\text{test}}| = 1000$ tests are used for evaluation. In each test, we employ the trapezoidal rule to simulate the PEC for $T_H = 0.01$ s with a time step of 0.1 μs, and the simulation data of the system (e.g., output voltage, inductor current, and fluctuation of duty cycle) is adopted to calculate the fitness value [4]. The three indexes corresponding to f_1, f_2, and f_3 are calculated as

$$f_1(X) = \frac{1}{T_H \cdot |S_{\text{test}}|} \sum_{c \in S_{\text{test}}} \int_0^{T_H} |u_{\text{out}}(t; c; X) - u^*_{\text{out}}| dt \tag{17}$$

$$f_2(X) = \frac{1}{T_H \cdot |S_{\text{test}}|} \sum_{c \in S_{\text{test}}} \int_0^{T_H} \max\big\{i_L(t; c; X) - i^*_L, 0\big\} dt \tag{18}$$

$$f_3(X) = \frac{1}{T_H \cdot |S_{\text{test}}|} \sum_{c \in S_{\text{test}}} \int_0^{T_H} |d(t; c; X) - d(t - T; c; X)| dt \tag{19}$$

where X is the parameter setting of the NNC, u^*_{out} is the output-voltage set point, i^*_L is the current limit, d is the duty cycle, and T is the switching period. Notably, the system performance is measured on the results (i.e., a control signal) produced by the NNC rather than on the NN parameters directly. The control signal actually can be expressed as a function of X, and hence, the parameter setting X of the NNC is included in (17)–(19) instead of the control signal. f_1 is to evaluate the stability of the output voltage, f_2 is to calculate the violation degree of the current, and f_3 is to evaluate the stability of the circuit. With the three indexes, the fitness function of system performance is defined as

$$\min F(X) = \phi_1 f_1(X) + \phi_2 f_2(X) + \phi_3 f_3(X) \tag{20}$$

where the factors ϕ_1, ϕ_2, and ϕ_3 represent the contributions of the corresponding indexes in the fitness evaluation. ϕ_1 and ϕ_3 are set as 1, and ϕ_2 is set as 100 to give a large penalty to the current limit violation according to [4]. By minimizing the fitness function, we can obtain the optimal NN parameters. Notably, formulas (17)–(19) with knowledge of the PCS are readily available, which can help the NN to learn the plant autonomously.

B. DE Optimization Procedure

Given the representation scheme and the objective function, the DE algorithm performs the following steps to search for the optimal solution for the controllers.

Step 1 (Initialization): Set $g = 0$. Each individual i initializes its solution X_i (the circuit components in an FNC or the network parameters in an NNC) by

$$x_{id} = L_d + \text{rand}(0, 1) \times (U_d - L_d) \tag{21}$$

where L_d and U_d are the low and upper bounds of the dth variable, respectively.

Step 2: Perform the mutation operator for each individual i as (1) to obtain the mutant vector V_i.

Step 3: Carry out the crossover operator for each individual i as (2) to obtain the vector U_i.

Step 4: Evaluate U_i for each individual I, and perform the selection operator as (3) to obtain the survived vector X_i. For the optimization of the PEC instance with an FNC, (6) is used for individual evaluation, and the one with a larger fitness value between U_i and X_i is selected to survive. For the optimization of the PEC instance with an NNC, (20) is used for individual evaluation, and the one with a smaller fitness value between U_i and X_i is selected.

Step 5: Check the termination condition. If the consumed fitness evaluation times or generations have reached the predefined maximum value, then go to Step 6; otherwise, set $g = g + 1$, and go back to Step 2 for the next generation.

Step 6: Output the best individual as the solution found.

C. RAS for DDE

As we know, controller optimization has expensive FEs that require a long computational time. This article extends the DE to DDE and dispatches individuals to different computing resources (e.g., a set of cores) for FE in Step 4. Assume

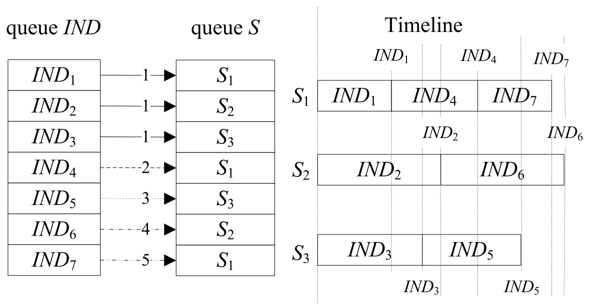

Fig. 3. Example of RAS to assign seven individuals IND_1–IND_7 to three resources S_1–S_3. The numbers on the lines between queue IND and queue S mean the corresponding assignment time. The timeline indicates the assigned individuals on each resource along time. The length of the rectangle with "IND_n" indicates the required computational time of IND_n on the corresponding resource.

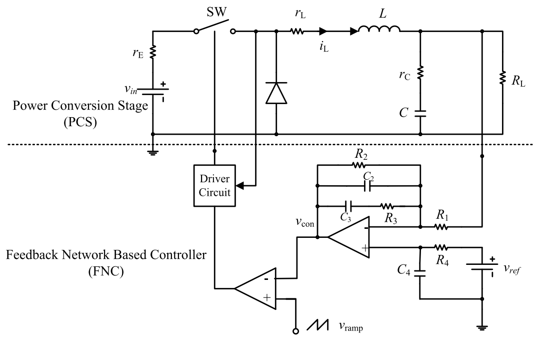

Fig. 4. Circuit schematic of the buck regulator.

that there are M computing resources S; then, one resource (i.e., S_0) is used for control, and the other $M-1$ resources (i.e., $S_1, \ldots, S_{M-1}$) are used for FE. The detailed assignment strategy RAS will be described in the following paragraph.

The distributed resources (i.e., cores in a cluster) often have different configurations and load burdens. Considering the time-varying characteristic of the resources, we do not build an explicit model to evaluate the computational ability and load state of each resource. Instead, we adopt the RAS to assign the individuals according to resources' real-time performance.

In RAS, the individuals and the resources are arranged in the queue IND and queue S, respectively. Once S is not empty, the assignment step is triggered. First, for each resource in S, it will be assigned one individual in IND, and then, it is removed from S. Note that, if an individual is assigned, it will also be removed from IND. Later, once a resource has finished the FE of the corresponding individual and returned the fitness value, it will be pushed back to queue S. The assignment procedure continues until IND becomes empty. In this way, the resources that have better performance on FE will be assigned more individuals, while the slow resources will be assigned fewer individuals. An example is given in Fig. 3.

The contribution of the proposed RAS is that it splits the evaluation process of all individuals into multiple jobs, and the multiple jobs are managed by a queue and assigned to multiple resources in a queue according to the real-time performance of the resources. Compared with the explicit-model-based methods [35], [36], the RAS has two advantages. On the one hand, no knowledge of the distributed resources needs to be known in advance (i.e., the computational time of the tasks), and no model for load assessment needs to be built. On the other hand, the assignment depends on the real-time performance of the resources but not a predictive model. This makes the assignment more accurate and better fit the time-varying feature of the distributed resources.

IV. PEC Design With FNC

A. Circuit Configurations

To evaluate the performance of RADDE and test its advantage in reducing computational time, we apply RADDE to optimize the PEC with an FNC, as shown in Fig. 4. It is a buck regulator with two parts, i.e., a classical buck converter as the PCS and a proportional-plus-integral controller as the FNC [37]–[39]. In the PCS part, we assume that R_L, r_C, and r_E are known in advance, and the components L and C are 200 and 1000 μF, respectively [37]. Since the PCS part always has a static characteristic, only the components R_1, R_2, R_3, R_4, C_2, C_3, and C_4 in FNC are optimized in this article. Particularly, the input voltage range v_{in} is set as [20 V, 60 V], the output load R_L is set in the range of [2 Ω, 12 Ω], the nominal output voltage is 5 V$\pm$1%, the switching frequency is 20 kHz, and the maximum ST is 20 ms. The search ranges for resistors R_1, R_2, R_3, and R_4 are set to be [100 Ω, 100 kΩ], and the search ranges for capacitors C_2, C_3, and C_4 are set to be [0.1 μF, 100 μF] following [39].

The fitness function (6) is used for evaluation. The parameters in (6) except the decision variables are set following [37]. Particularly, $\delta R_L = 3$, $\delta v_{\text{in}} = 20$, $N_s = 15\,000$, $N_T = 6$, $K_1 = 2$, $K_3 = 2$, $K_5 = 2$, $K_7 = 2$, $K_9 = 2$, $T_{s0} = 0.025$, $M_{p0} = 4$, and $M_{v0} = 4$ are user-defined parameters according to circuit knowledge and system characteristics; $K_2 = 400$, $K_4 = 0.455$, $K_6 = 0.455$, $K_8 = 2.28 \times 10^{-3}$, and $K_{10} = 32$ are determined according to (10)–(14); M_p, M_v, T_s, and A_1 are derived from the system when performing tests.

B. Algorithm Configurations

GA [37], particle swarm optimization (PSO) [40], and orthogonal learning PSO (OLPSO) [39] are taken for comparisons since they have been successfully applied to PEC optimization and are representative. In the GA, the population size is set as 30, the crossover rate is set as 0.85, and the mutation probability is set as 0.25. In the PSO and OLPSO, the inertia weight linearly decreases from 0.9 to 0.4, the acceleration coefficients c_1 and c_2 are set as 2.0, and the population size is set as 30 and 40, respectively. The above parameter settings are according to their original papers, which have been well-tuned for the PEC optimization. In the proposed RADDE, the population size is set as 100, and F and CR are set as 0.5 and 0.9, respectively, according to the suggestion of [42] and [43]. For fair comparisons, all the algorithms use a maximal FEs of 1.5×10^4. The experiment is carried out 30 times for each algorithm, and the statistical results are reported. For clarity, the results of the best algorithm are marked in **boldface**.

TABLE I
EXPERIMENTAL RESULT COMPARISONS OF DIFFERENT APPROACHES

Approaches	Median	Std. Dev	Best	Worst	Success #	Wilcoxon test
GA	110.745	10.0350	131.027	97.0136	0	Z=4.77185†
PSO	143.756	24.5561	192.682	112.021	13	Z=3.84628†
OLPSO	184.497	18.6530	192.961	137.863	**25**	Z=0.94614
RADDE	**192.635**	**22.1625**	**193.020**	**137.897**	**25**	NA

†The difference is significant by the Wilcoxon's rank sum test at a 0.05 significance level. The 'Success #' is the count of runs that can find final solution with a fitness larger than 150.

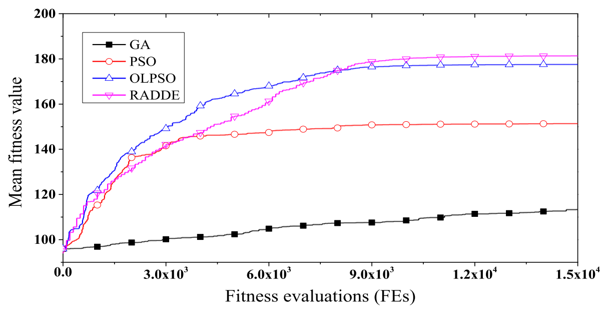

Fig. 5. Convergence curves of different methods for optimizing the FNC.

C. Evaluation on Solution Quality

From Table I, we can see that RADDE obtains the best "median," "best," and "worst" values. According to Wilcoxon's rank-sum test at a 0.05 significance level, RADDE performs significantly better than other algorithms. In addition, given an acceptable fitness value of 150 following the suggestion of Zhan and Zhang [39], RADDE succeeded in 25 out of the 30 runs, showing its strong reliability. Although RADDE has similar best and worst values to OLPSO, RADDE obtained 16 runs (which is more than half of the 30 runs) with fitness values larger than 192.632 and five runs below 150, resulting in a median value close to the best and a large Std Dev. From the convergence curves illustrated in Fig. 5, it can be seen that the proposed RADDE surpasses other algorithms and converges to better results. The better performance of RADDE mainly benefits from its strong exploration ability. Particularly, both PSO and OLPSO only learn from a particle's own history and the global/neighboring best particle, while RADDE constructs new solutions using a difference component of two solutions randomly selected from the whole population [as shown in (1)], which provides stronger exploration ability to avoid being trapped in local optima. In addition, the crossover operator enables RADDE to directly inherit some dimensions of good solutions, helping accelerate algorithm convergence.

To observe the response of the circuit under the disturbances of voltage and load, the best solutions obtained by different algorithms are taken as the component values of FNC to carry out the simulations. In the beginning, the input voltage v_{in} is 20 V, and the output load R_L is 5 Ω. Then, the input voltage is suddenly changed in 30 ms, and the load is suddenly changed in 60 ms. The setting time and disturbance responses of the voltage and current responded by the FNC obtained by different algorithms are illustrated in Figs. 6 and 7, respectively. We can see that the proposed RADDE presents a smaller overshoot in the inductor current and voltage and requires a shorter ST to reach a steady state. RADDE is able to optimize the circuit to achieve better dynamic performance and stronger responseability.

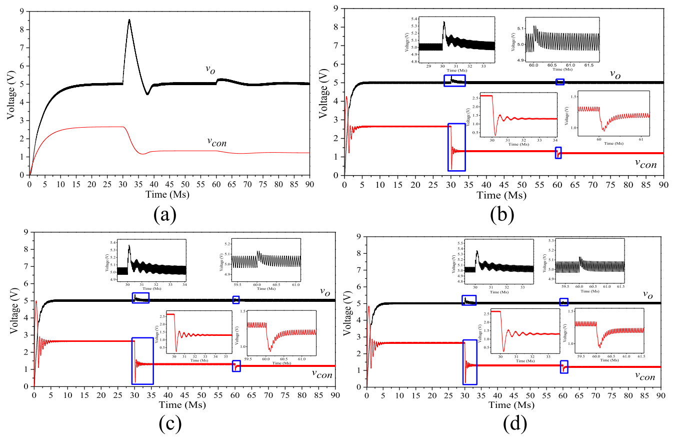

Fig. 6. Simulated voltage responses. From 0 to 30 ms, v_{in} is 20 V, and R_L is 5 Ω; in 30 ms, v_{in} is suddenly changed from 20 to 40 V; and in 60 ms, R_L is suddenly changed from 5 to 10 Ω. The curves framed by blue boxes are enlarged. (a) GA. (b) PSO. (c) OLPSO. (d) RADDE.

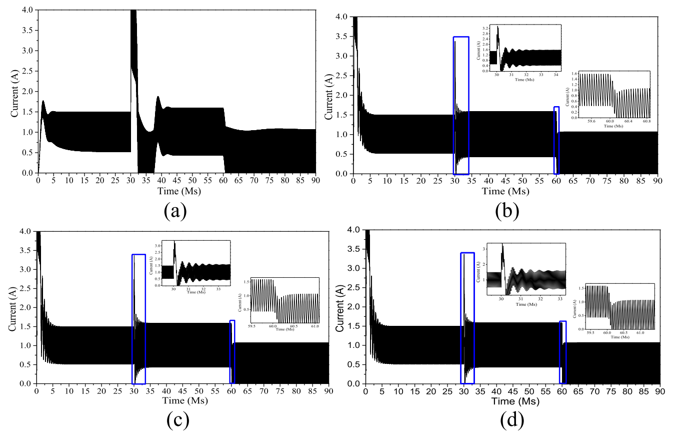

Fig. 7. Simulated current responses. From 0 to 30 ms, v_{in} is 20 V, and R_L is 5 Ω; in 30 ms, v_{in} is suddenly changed from 20 to 40 V; and in 60 ms, R_L is suddenly changed from 5 to 10 Ω. The curves framed by blue boxes are enlarged. (a) GA. (b) PSO. (c) OLPSO. (d) RADDE.

TABLE II
FIVE CONFIGURATIONS IN HOMOGENEOUS ENVIRONMENT

No.	Configuration of the machine			Number of machines (n)	Number of cores (Ss) used totally (M)
	CPU	Frequency	Cores		
C1	Intel Xeon Phi™ CPU 7250	1.40GHz	68	1	68
C2	Intel® core™ i7-7700 CPU	3.60GHz	4	6	21
C3	Intel® core™ i5-7500 CPU	3.40GHz	4	6	21
C4	Intel® Xeon® CPU E3-1225 v5	3.30GHz	4	6	21
C5	Intel® Xeon® CPU E5-1603 v4	2.80GHz	4	2	8

D. Acceleration in Computational Time of RADDE

Since FE is the most time-consuming part, the computational time of GA, PSO, and OLPSO is similar, which is about 829.185 s on Intel Core i7-7700 CPU with 3.60 GHz. The parallel RADDE is speeded up under multiresource envi-

TABLE III
COMPUTATIONAL TIME (SECONDS) OF DIFFERENT ASSIGNMENT STRATEGIES ON CONFIGURATIONS C1–C5 WITH DIFFERENT LOAD STATES IN HOMOGENEOUS ENVIRONMENTS

Load case	Configuration	The Cores (Rs) in Different load states					Number of cores (Rs)	Independent run (IR)			Simultaneous run (SR)		
		Idle	Low	Middle	High	Very high		UAS	CAS	RAS	UAS	CAS	RAS
L_0	C1	S_0-S_{67}	-*	-	-	-	68	**224.049**	294.987	244.261	353.272	399.097	**350.497**
	C2	S_0-S_{20}	-	-	-	-	21	**75.38**	118.667	93.753	287.749	337.449	**167.39**
	C3	S_0-S_{20}	-	-	-	-	21	**79.699**	125.050	100.46	311.702	355.398	**170.02**
	C4	S_0-S_{20}	-	-	-	-	21	**84.764**	131.165	102.053	330.65	374.103	**177.309**
	C5	S_0-S_7	-	-	-	-	8	**302.889**	334.742	324.0374	1036.16	1105.04	**621.159**
L_1	C1	-	-	-	S_0-S_{67}	-	68	**					
	C2	S_0-S_{16}	-	-	S_{17}-S_{20}	-	21	497.445	149.054	**117.761**	703.287	739.019	**216.847**
	C3	S_0-S_{16}	-	-	S_{17}-S_{20}	-	21	546.566	156.517	**128.169**	783.413	814.494	**235.715**
	C4	S_0-S_{16}	-	-	S_{17}-S_{20}	-	21	584.005	164.881	**135.715**	827.067	862.588	**249.198**
	C5	S_0-S_3	-	-	S_4-S_7	-	8	1819.34	569.703	**557.648**	2619.48	2721.11	**935.025**
L_2	C1	-	-	-	-	-	68						
	C2	S_0-S_{12}	-	-	S_{13}-S_{20}	-	21	502.766	179.251	**132.654**	714.14	747.10	**220.03**
	C3	S_0-S_{12}	-	-	S_{13}-S_{20}	-	21	558.685	187.643	**141.139**	794.759	829.578	**238.927**
	C4	S_0-S_{12}	-	-	S_{13}-S_{20}	-	21	594.512	198.477	**141.673**	842.878	875.147	**255.413**
	C5	-	-	-	S_0-S_7	-	8	2108.93	1907.21	**1235.126**	2592.555	2724.08	**1536.58**
L_3	C1	-	-	-	-	-	68						
	C2	S_0-S_4	-	-	S_5-S_{20}	-	21	504.995	419.695	**230.614**	732.611	747.192	**303.706**
	C3	S_0-S_4	-	-	S_5-S_{20}	-	21	559.254	441.218	**250.665**	797.542	829.903	**318.546**
	C4	S_0-S_4	-	-	S_5-S_{20}	-	21	593.073	473.486	**267.414**	845.782	879.337	**342.096**
	C5	-	-	-	-	-							
L_4	C1	-	-	-	-	-	68						
	C2	S_0-S_4	S_5-S_8	S_9-S_{12}	S_{13}-S_{16}	S_{17}-S_{20}	21	646.359	420.008	**216.122**	850.794	883.680	**291.275**
	C3	S_0-S_4	S_5-S_8	S_9-S_{12}	S_{13}-S_{16}	S_{17}-S_{20}	21	717.975	442.024	**234.519**	957.332	993.746	**314.884**
	C4	S_0-S_4	S_5-S_8	S_9-S_{12}	S_{13}-S_{16}	S_{17}-S_{20}	21	769.771	467.086	**245.82**	1004.5	1040.4	**342.033**
	C5	-	-	-	-	-	8						

* THE CONFIGURATION IS NOT CONSIDERED. ** THE RESULTS ARE NOT AVAILABLE DUE TO THE LONG TIME

ronments. This section investigates the computational time acceleration of RADDE in different environments (i.e., distributed resources with different computing abilities and load states). The typical uniform assignment strategy (UAS) [32] (i.e., individuals are dispatched to resources evenly) and the CPU-load-based assignment strategy (CAS) [35] (i.e., individuals are dispatched to resources based on resource load) are also integrated into DDE to compare with RADDE. For fair comparisons, we, on the one hand, carry out an independently run (IR) for each algorithm in each environment. Moreover, we, on the other hand, carry out simultaneously run (SR), where all the three algorithms run in each environment at the same time to observe their behaviors in the same running environment. Notably, since UAS, CAS, and RAS are only used for resource assignment, they do not influence the performance of the optimization algorithms in terms of fitness evaluation but work differently in terms of computational time. Experiments are performed in both homogeneous and heterogeneous environments.

1) Homogeneous Environment: Experiments are independently carried out in five homogenous environments named from C1 to C5, as listed in Table II. In each configuration environment, five kinds of resource-load cases named L_0–L_4 are designed for test and are listed in Table III. The resource-load cases are created to make the homogeneous resources different; hence, resource selection is meaningful. They also fit the real scenes in distributed platforms where the available resources usually have supported applications (loads) from other users. These specific configurations help comprehensively verify the effect of the resource assignment strategies under different resource environments. Each core can be in idle-, low-, medium-, high-, and very high-load states. The different load states can be obtained by running some special processes on the CPU. From L_0 to L_3, the number of resources in high-load state increases, and the computing resources become tighter. In the simulations, all the cores in C1 and C5 are used. Specifically, in C2–C4, to make the population size divisible by the number of cores, only 21 cores are used.

TABLE IV
CONFIGURATION IN HETEROGENEOUS ENVIRONMENT

No.	Configuration of each machine			n	Name	M
	CPU	Frequency	Cores			
C6	Intel® core™ i7-7700 CPU	3.60GHz	4	2	S_0-S_7	24
	Intel® core™ i5-7500 CPU	3.40GHz	4	1	S_8-S_{11}	
	Intel® Xeon® CPU E3-1225 v5	3.30GHz	4	1	S_{12}-S_{15}	
	Intel® Xeon® CPU E5-1603 v4	2.80GHz	4	2	S_{16}-S_{23}	

The results are reported in Table III. We can see that the proposed RAS obtains the best values (the shortest computational time) on almost all cases, especially on the high-load cases, for example, on the cases of IR with L_1–L_4 and on all cases of SR. These indicate that RAS can distinguish the different load states of the resources and assign individuals for load balance to obtain the shortest computational time. Specifically, the UAS obtains a slightly shorter time than RAS in the cases of IR with L_0. This is because all the cores have the same computing ability and load state, and therefore, a uniform assignment is the best choice. In contrast, the CAS

TABLE V
COMPUTATIONAL TIME (SECONDS) OF ASSIGNMENT STRATEGIES ON CONFIGURATION C6 WITH DIFFERENT LOAD STATES IN HETEROGENEOUS ENVIRONMENT

Load case	Configuration	Cores (Ss) in Different states					Number of cores (Ss)	Independent run (IR)			Simultaneously run (SR)		
		Idle	Low	Middle	High	Very high		UAS	CAS	RAS	UAS	CAS	RAS
L_0	C6	S_0-S_{23}	-	-	-	-	24	**82.290**	129.666	97.413	311.403	358.351	**157.816**
L_1	C6	S_0-S_3,S_8-S_{23}	-	-	S_4-S_7	-	24	569.425	**147.457**	155.762	815.263	855.621	**197.471**
L_2	C6	S_0-S_3,S_{12}-S_{23}	-	-	S_4-S_{11}	-	24	565.657	164.767	**155.910**	796.967	839.757	**243.490**
L_3	C6	S_0-S_3,S_{20}-S_{23}	-	-	S_4-S_{19}	-	24	569.452	284.841	**199.144**	835.441	874.673	**297.919**
L_4	C6	S_0-S_4	S_5-S_8	S_9-S_{12}	S_{13}-S_{16}	S_{17}-S_{23}	24	593.609	284.778	**196.221**	783.527	826.341	**264.543**

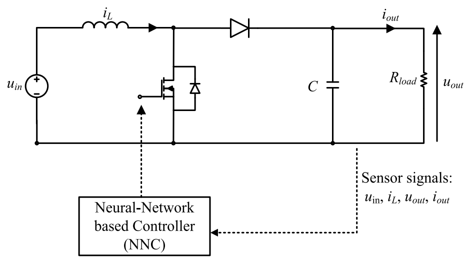
i_L
i_{out}
u_{in}
C
R_{load}
u_{out}
Sensor signals:
u_{in}, i_L, u_{out}, i_{out}
Neural-Network based Controller (NNC)

Fig. 8. Circuit schematic of the boost converter.

TABLE VI
SPECIFICATIONS OF THE BOOST CONVERTER

Description	Parameter	Value
Input voltage	u_{in}	12-20V
Output voltage reference	u_{out}*	48V
Maximum output current	i_{out}	2.5A
Inductor current limit	i_L*	15A
Inductor	L	160μH
Output capacitor	C	470μF
Switching frequency	f_{sw}	100kHz

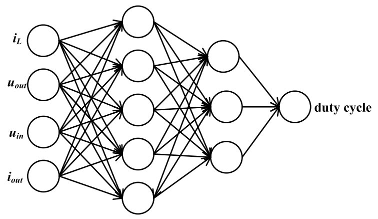

Fig. 9. Structure of the NN for the controller.

performs generally better than UAS in the IR cases because it can judge the idle or busy states of different cores and assign more individuals to idle cores to reduce time. However, the CAS may fail on distinguishing the different degrees of the load states for busy resources. Therefore, it is generally outperformed by both UAS and RAS in the SR cases.

2) Heterogeneous Environment: In this section, we test the adaptive ability of the RAS to the heterogeneous environment. The configuration named C6 is listed in Table IV. The results on C6 with different load cases are reported in Table V. RAS obtains the shortest time on eight out of ten cases. Specifically, in the case of IR with L_0, the computational time of RAS is slightly larger than UAS mainly due to the balance between the transmission time and the discrepancy of the computational ability of the resources. The RAS costs more on transmission since it assigns the individuals one by one. However, the advantage of RAS is obvious in the cases in which different resources have different load states, i.e., the cases of IR with L_2–L_4 and all cases of SR. For CAS, it obtains a slightly shorter time than RAS in the IR of L_1 but fails on all the other cases since it can judge the load but not the computational abilities of different resources. Thus, CAS does better than UAS in IR cases but worse in SR cases, while RAS does best to utilize the resources more effectively in both cases.

V. PEC DESIGN WITH NNC

A. Circuit Configurations

In this section, we take a very recent boost converter with NNC as an example to observe the effectiveness of the proposed RADDE in NN training. The circuit schematic of the boost converter is shown in Fig. 8. The NNC receives four sensor signals, including the inductor current i_L, the output voltage u_{out}, the input voltage u_{in}, and the output current i_{out}, from the PCS, and then outputs the duty cycle for the main switch. The structure of the NN is illustrated in Fig. 9. There are one input layer with four neurons, two hidden layers with five and three neurons, respectively, and one output layer with one neuron. i_L, u_{out}, u_{in}, and i_{out} are the inputs of the NN, and the duty cycle is the output of the NN. Herein, the duty cycle is a signal to control the circuit in PEC. In the NN, the output f_k of the kth neuron in the input layer is directly the same as its input value. For the following layers, the output $f_{i,j}$ for the jth neuron in layer i is calculated as:

$$f_{i,j} = \varphi\left(\sum_k w_{i-1,k,i,j} f_{i-1,k} + b_{i,j}\right) \tag{22}$$

where $w_{i-1,k,i,j}$ is the weight of the connection between the kth neuron in layer $i-1$ and the jth neuron in layer i, $b_{i,j}$ is the bias of the jth neuron in layer i, and φ is the activation function. Herein, the activation function is the sigmoid function as

$$\varphi(x) = \frac{1}{1+e^{-x}}. \tag{23}$$

Due to the physical constraints, the output duty cycle of the NN is limited below 0.8. The ranges of the parameters (connection weights and biases) in NN are set as [−2000, 2000] following [4]. The required circuit specifications are described in Table VI according to [4].

TABLE VII
RESULTS OF DDE AND DPSO ON OPTIMIZING NNC

Algorithms	FEs	Best fitness value	Computational time	Ratio of time
UAS/DPSO	7,680,000	0.86602	894307.0s	17.59
RAS/DPSO	7,680,000	0.86602	278712.0s	5.48
UAS/DDE	800,000	**0.62737**	106278.0s	2.09
RADDE	800,000	**0.62737**	**50844.2s**	**1.00**

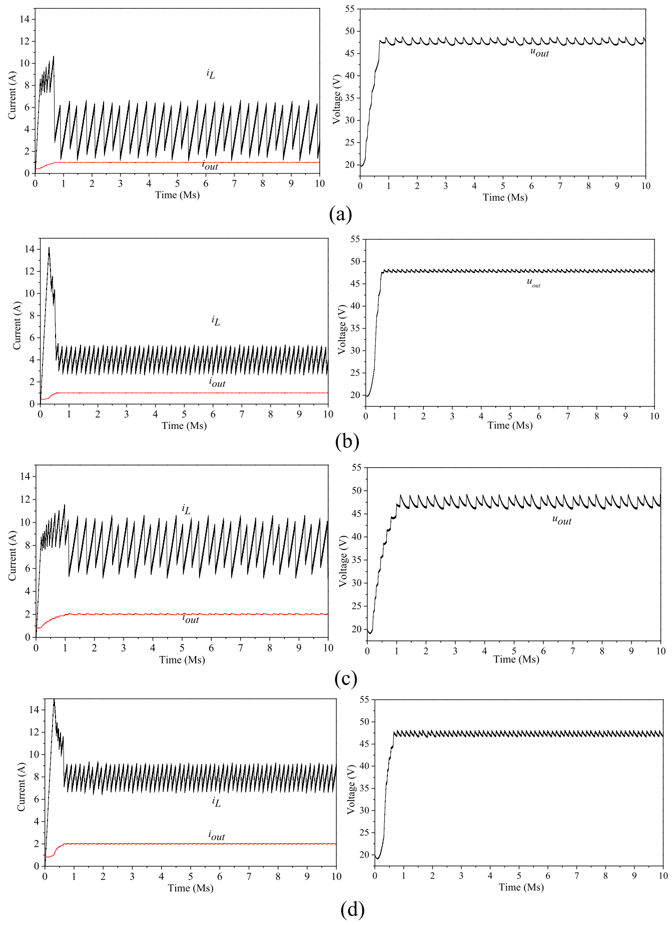

Fig. 10. Startup transient performance comparison. The change in the inductor current, output current, and output voltage of the boost converter under R_{load} of (a) and (b) 47.5 and (c) and (d) 23.5 Ω. (a) Boost converter with NNC optimized by DPSO, $R_{\text{load}} = 47.5$ Ω. (b) Boost converter with NNC optimized by RADDE, $R_{\text{load}} = 47.5$ Ω. (c) Boost converter with NNC optimized by DPSO, $R_{\text{load}} = 23.5$ Ω. (d) Boost converter with NNC optimized by RADDE, $R_{\text{load}} = 23.5$ Ω.

B. Environment and Algorithm Configurations

In the experiment, we apply the proposed RADDE to the NNC optimization on a cluster. The cluster includes one server with 68 cores and 1.40 GHz, eight machines with eight cores and 3.60 GHz, eight machines with eight cores and 3.40 GHz, and four machines with eight cores and 3.30 GHz. Among the total 228 cores, one core is used for the master node, and the other 227 cores are used for slave nodes.

The population size of RADDE is set as 400 following the suggestions of Storn and Price [42] and Zhang and Sanderson [43], and the maximum generation is set as 2000 following [4]. The distributed PSO (DPSO) for NN training in [4] is taken for comparison. The population size of DPSO is set as 3840, and the maximum generation is set as 2000 following the original paper [4]. Thus, DPSO uses a maximum FEs of 7 680 000, while the proposed DDE adopts only 800 000 FEs. Note that the backpropagation method is not compared since a large training data set is hard to construct for the PEC. The expected control signal of the NNC is unknown for a given input. To test the effectiveness of the proposed RAS, we also integrate the DPSO with UAS and RAS, forming UAS/DPSO and RAS/DPSO, respectively. Therefore, four algorithms, UAS/DPSO, RAS/DPSO, UAS/DDE, and RADDE (i.e., the RADDE), are performed for comparisons. Since the execution time is very long, each algorithm only runs one time, and the obtained result and computational time in seconds are reported.

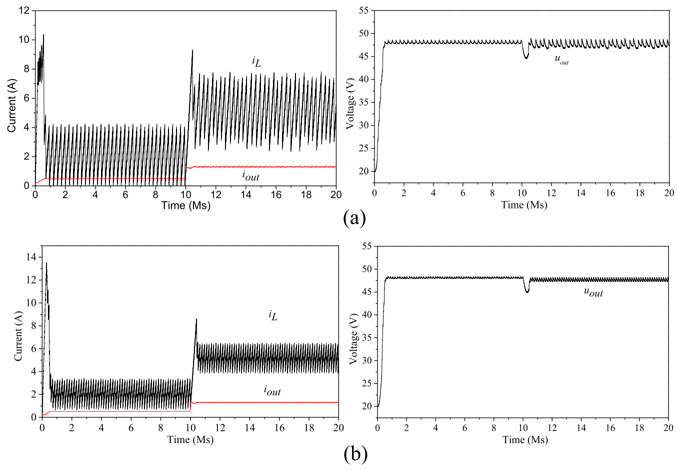

Fig. 11. Simulated current and voltage responses from 0 to 20 ms. u_{in} is 12 V. From 0 to 10 ms, R_{load} is 94 Ω; in 10 ms, R_{load} is suddenly changed from 94 to 36.5 Ω. (a) Boost converter with NNC optimized by DPSO. (b) Boost converter with NNC optimized by RADDE.

C. Result and Comparison

The fitness value [see (20)] of the best solution found and the computational time of DPSO and DDE are reported in Table VII. It can be seen that the proposed DDE has a stronger global searchability. Compared with the DPSO, our proposed DDE uses fewer FEs to obtain a better value of 0.62737.

For the computational time, the RADDE gets the shortest time. Comparing UAS and RAS, the RAS requires only nearly a half of the time of UAS for DDE, while it reduces a ratio of 68.8% to UAS on time for DPSO. For example, the computational time of RADDE (50 844.2 s) is much shorter than that of UAS/DDE (106 278.0 s). Similarly, the UAS/DPSO requires 894 307.0 s, while RAS/DPSO needs only 278 712.0 s. It indicates that the training time for UAS/DPSO is even more than ten days, which seems impractical. We also set the computational time of RADDE as unit 1.00 and calculate the ratio of time required by other algorithms compared with RADDE. From the ratio of the time required by RAS to UAS in DDE and DPSO, we can also see that the more individuals in one generation (DPSO uses more individuals than DDE), the more obvious the acceleration obtained by the RAS (17.59/5.48 > 2.09/1.00). Moreover, the acceleration of UAS is limited by the slowest resources. In contrast, the RAS can assign the individuals according to the computational ability of the resources to ease the limitation of the slow resources as much as possible. In this way, more individuals are assigned to the fast resources, while fewer individuals are assigned to the slow resources so that the resources can be utilized efficiently. In general, the proposed RADDE can obtain a better result and significantly reduce the computational time on NNC optimization.

D. Comparisons on Simulation Results

To test the startup transient performance of the trained NNC, two boost converters with NNC optimized by the DPSO and RADDE, respectively, are taken for simulation. The input voltage is set as 12 V, and the load is set as two values of $R_{\text{load}} = 47.5\ \Omega$ and $R_{\text{load}} = 23.5\ \Omega$. The inductor current, the output current, and the output voltage under these two loads are reported in Fig. 10. We can see that the NNC optimized by RADDE performs better. It presents a faster setting and smaller voltage overshoot under different R_{load}'s. Moreover, both the output current and inductor current are strictly below the maximum limit.

To further observe the transient response of the two NNCs under large load disturbance, we perform a simulation with large signal disturbances for 20 ms. In 0–10 ms, the input voltage is 12 V, and the load R_{load} is 94 Ω; then, the load R_{load} is suddenly changed to 36.5 Ω in 10 ms. As demonstrated in Fig. 11, we can see that the NNC optimized by RADDE has a stable response for load disturbance. For example, in 10 ms, although both the two NNCs respond to the load change quickly, the one optimized by RADDE presents smaller current disturbance and output voltage overshoot. The boost converter with RADDE-optimized NNC reaches the set point much faster.

Generally, the RADDE-optimized NNC achieves better performance on startup settling and fast/stable response on large-signal disturbances. Therefore, the proposed RADDE performs better than DPSO on NNC optimization.

VI. Conclusion

For the modern NNC optimization in PEC, solution quality and computational time are two challenging issues. This article develops a RADDE algorithm that the DDE is for optimization and the RAS is for computational time acceleration. Individuals are adaptively dispatched for FE based on the real-time performance of the resources.

The performance of the RADDE algorithm has been verified on two PEC instances, including FNC optimization for a buck regulator and NNC optimization for a boost converter. Experimental results show that our proposed RADDE can obtain better results on both FNC and NNC optimization within significantly shorter computational time. By distinguishing the computational ability and the load states of different resources, the adaptive dispatch mode of RAS is able to ease the limitation of the slow resources and reduce the execution time as far as possible. In general, the proposed RADDE is competitive to other EC algorithms on both solution quality and computational time for controller optimization. This also provides a new promising approach for ENN. In the future, we will apply the proposed method on more sophisticated controllers in PEC and look deep into the exploration and exploitation abilities of the proposed method.

References

[1] Z. Huang and V. Dinavahi, "A fast and stable method for modeling generalized nonlinearities in power electronic circuit simulation and its real-time implementation," *IEEE Trans. Power Electron.*, vol. 34, no. 4, pp. 3124–3138, Apr. 2019.

[2] M. A. Elsaharty, A. Luna, J. I. Candela, and P. Rodriguez, "A unified power flow controller using a power electronics integrated transformer," *IEEE Trans. Power Del.*, vol. 34, no. 3, pp. 828–839, Jun. 2019.

[3] T. Dragicevic, "Model predictive control of power converters for robust and fast operation of AC microgrids," *IEEE Trans. Power Electron.*, vol. 33, no. 7, pp. 6304–6317, Jul. 2018.

[4] W. Wang *et al.*, "Training neural-network-based controller on distributed machine learning platform for power electronics systems," in *Proc. IEEE Energy Convers. Congr. Expo. (ECCE)*, Oct. 2017, pp. 3083–3089.

[5] J. P. Larue, "A bi-directional neural network based on a convolutional neural network and associative memory matrices that meets the universal approximation theorem," Jadco Signals, Charleston, SC, USA, Tech. Rep. 13157179009.

[6] J. P. Larue, G. Ioup, J. Ioup, and G. Smith, "Channel order selection in blind deconvolution based on eigenvector characteristics and using normal modes in conjunction with multipath compression for source identification," *J. Acoust. Soc. Amer.*, vol. 113, no. 4, p. 2212, 2003.

[7] H. Xu and S. Jagannathan, "Neural network-based finite horizon stochastic optimal control design for nonlinear networked control systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 3, pp. 472–485, Mar. 2015.

[8] T. Wang, H. Gao, and J. Qiu, "A combined adaptive neural network and nonlinear model predictive control for multirate networked industrial process control," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 2, pp. 416–425, Feb. 2016.

[9] X.-F. Liu *et al.*, "Neural network-based information transfer for dynamic optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 5, pp. 1557–1570, May 2020.

[10] N. Qin, K. Liang, D. Huang, L. Ma, and A. H. Kemp, "Multiple convolutional recurrent neural networks for fault identification and performance degradation evaluation of high-speed train bogie," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 12, pp. 5363–5376, Dec. 2020.

[11] R. Q. Fuentes-Aguilar and I. Chairez, "Adaptive tracking control of state constraint systems based on differential neural networks: A barrier Lyapunov function approach," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 12, pp. 5390–5401, Dec. 2020.

[12] S. Maiti, V. Verma, C. Chakraborty, and Y. Hori, "An adaptive speed sensorless induction motor drive with artificial neural network for stability enhancement," *IEEE Trans. Ind. Informat.*, vol. 8, no. 4, pp. 757–766, Nov. 2012.

[13] M. Kaminski and T. Orlowska-Kowalska, "Adaptive neural speed controllers applied for a drive system with an elastic mechanical coupling—A comparative study," *Eng. Appl. Artif. Intell.*, vol. 45, pp. 152–167, Oct. 2015.

[14] X. Zhan, W. Wang, and H. Chung, "A neural-network-based color control method for multi-color LED systems," *IEEE Trans. Power Electron.*, vol. 34, no. 8, pp. 7900–7913, Aug. 2019.

[15] J. de Jesús Rubio, P. Angelov, and J. Pacheco, "Uniformly stable backpropagation algorithm to train a feedforward neural network," *IEEE Trans. Neural Netw.*, vol. 22, no. 3, pp. 356–366, Mar. 2011.

[16] A. R. Heravi and G. A. Hodtani, "A new correntropy-based conjugate gradient backpropagation algorithm for improving training in neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 12, pp. 6252–6263, Dec. 2018.

[17] Z.-C. Fan, T.-S.-T. Chan, Y.-H. Yang, and J.-S.-R. Jang, "Backpropagation with N-D vector-valued neurons using arbitrary bilinear products," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 7, pp. 2638–2652, Jul. 2020.

[18] G. Nopoles, F. Vanhoenshoven, R. Falcon, and K. Vanhoof, "Nonsynaptic error backpropagation in long-term cognitive networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 3, pp. 865–875, Mar. 2020.
[19] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Trans. Evol. Comput.*, vol. 23, no. 5, pp. 828–841, Oct. 2019.
[20] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 601–614, Feb. 2019.
[21] Y. Zhou, G. G. Yen, and Z. Yi, "Evolutionary compression of deep neural networks for biomedical image segmentation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 8, pp. 2916–2929, Aug. 2020.
[22] M. Nadji-Tehrani and A. Eslami, "A brain-inspired framework for evolutionary artificial general intelligence," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 12, pp. 5257–5271, Dec. 2020.
[23] C. Zhang, K. C. Tan, H. Li, and G. S. Hong, "A cost-sensitive deep belief network for imbalanced classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 1, pp. 109–122, Jan. 2019.
[24] Y. Sun, G. G. Yen, and Z. Yi, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 89–103, Feb. 2019.
[25] R. Storn and K. Price, "Differential evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces," Int. Comput. Sci. Inst., Berkeley, CA, USA, Tech. Rep. TR-95-012, 1995.
[26] Z.-H. Zhan *et al.*, "Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 704–716, Mar. 2017.
[27] C.-H. Chen and C.-B. Liu, "Reinforcement learning-based differential evolution with cooperative coevolution for a compensatory neuro-fuzzy controller," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4719–4729, Oct. 2018.
[28] Z.-J. Wang *et al.*, "Dual-strategy differential evolution with affinity propagation clustering for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, vol. 22, no. 6, pp. 894–908, Dec. 2018.
[29] Z.-J. Wang *et al.*, "Automatic niching differential evolution with contour prediction approach for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, vol. 24, no. 1, pp. 114–128, Feb. 2020.
[30] Z.-G. Chen, Z.-H. Zhan, H. Wang, and J. Zhang, "Distributed individuals for multiple peaks: A novel differential evolution for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, vol. 24, no. 4, pp. 708–719, Aug. 2020.
[31] A. Mondal, S. Ghosh, and A. Ghosh, "Distributed differential evolution algorithm for MAP estimation of MRF model for detecting moving objects," in *Proc. Int. Conf. Image Inf. Process.*, Nov. 2011, pp. 1–6.
[32] S. Santander-Jiménez and M. A. Vega-Rodríguez, "Parallel multiobjective metaheuristics for inferring phylogenies on multicore clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1678–1692, Jun. 2015.
[33] A. Mosaddegh, C. A. Cañizares, K. Bhattacharya, and H. Fan, "Distributed computing architecture for optimal control of distribution feeders with smart loads," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1469–1478, May 2017.
[34] X.-F. Liu, Z.-H. Zhan, J.-H. Lin, and J. Zhang, "Parallel differential evolution based on distributed cloud computing resources for power electronic circuit optimization," in *Proc. Genet. Evol. Comput. Conf. Companion*, Jul. 2016, pp. 117–118.
[35] N. Ma, X.-F. Liu, Z.-H. Zhan, J.-H. Zhong, and J. Zhang, "Load balance aware distributed differential evolution for computationally expensive optimization problems," in *Proc. Genet. Evol. Comput. Conf. Companion*, Jul. 2017, pp. 209–210.
[36] J. M. Arroyo and A. J. Conejo, "A parallel repair genetic algorithm to solve the unit commitment problem," *IEEE Trans. Power Syst.*, vol. 17, no. 4, pp. 1216–1224, Nov. 2002.
[37] J. Zhang, H. S. H. Chung, W.-L. Lo, S. Y. Hui, and A. K.-M. Wu, "Implementation of a decoupled optimization technique for design of switching regulators using genetic algorithms," *IEEE Trans. Power Electron.*, vol. 16, no. 6, pp. 752–763, Nov. 2001.
[38] B. K. H. Wong and H. S.-H. Chung, "Steady-state analysis of PWM DC/DC switching regulators using iterative cycle time-domain simulation," *IEEE Trans. Ind. Electron.*, vol. 45, no. 3, pp. 421–432, Jun. 1998.
[39] Z. H. Zhan and J. Zhang, "Orthogonal learning particle swarm optimization for power electronic circuit optimization with free search range," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2011, pp. 2563–2570.
[40] M. A. Rahman, S. Anwar, and A. Izadian, "Electrochemical model parameter identification of a lithium-ion battery using particle swarm optimization method," *J. Power Sources*, vol. 307, pp. 86–97, Mar. 2016.
[41] X.-F. Liu *et al.*, "Historical and heuristic-based adaptive differential evolution," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 12, pp. 2623–2635, Dec. 2019.
[42] R. Storn and K. Price, "Differential evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces," Int. Comput. Sci. Inst., Berkeley, CA, USA, Tech. Rep. TR-95-012, 1995. [Online]. Available: http://http.icsi.berkeley.edu/~storn/litera.html
[43] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, Oct. 2009.

Xiao-Fang Liu (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Sun Yat-sen University, Guangzhou, China, in 2015 and 2020, respectively.

She is currently a Lecturer with the College of Artificial Intelligence, Nankai University, Tianjin, China. Her current research interests include evolutionary computation, neural networks, and their applications in design and optimization, such as cloud computing resource scheduling and multirobot systems.

Zhi-Hui Zhan (Senior Member, IEEE) received the bachelor's and Ph.D. degrees from the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, in 2007 and 2013, respectively.

He is currently the Changjiang Scholar Young Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His current research interests include evolutionary computation algorithms, neural networks, and power electronic circuits.

Dr. Zhan's doctoral dissertation was awarded the IEEE Computational Intelligence Society (CIS) Outstanding Ph.D. Dissertation and the China Computer Federation (CCF) Outstanding Ph.D. Dissertation. He was a recipient of the Outstanding Youth Science Foundation from the National Natural Science Foundation of China (NSFC) in 2018 and the Wu Wen Jun Artificial Intelligence Excellent Youth from the Chinese Association for Artificial Intelligence in 2017. He is listed as one of the Highly Cited Chinese Researchers in Computer Science. He is also an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and *Neurocomputing*.

Jun Zhang (Fellow, IEEE) received the Ph.D. degree from the City University of Hong Kong, Hong Kong, in 2002.

He is currently a Visiting Scholar with Zhejiang Normal University, Jinhua, China, and the Chaoyang University of Technology, Taichung, Taiwan. His current research interests include computational intelligence and power electronic circuits.

Dr. Zhang was a recipient of the Changjiang Chair Professor from the Ministry of Education, China, in 2013, and the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011. He is also an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON CYBERNETICS, and the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS.