IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 15, 2020, accepted April 27, 2020, date of publication May 25, 2020, date of current version September 4, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2997037

An FRTDS Real-Time Simulation Optimized Task Scheduling Algorithm Based on Reinforcement Learning

Y. GUAN, BD. ZHANG, AND Z. JIN
Electrical Engineering Department, Tianjin University, Tianjin 300072, China

Corresponding author: Y. Guan (hoffman_g_0601@163.com)

This work was supported by the Key Laboratory of Smart Grid of Ministry of Education, Tianjin University, Tianjin, China.

ABSTRACT This paper presents a deep reinforcement learning (DRL)-based task scheduling algorithm that is applied to an FPGA-based real-time digital simulation (FRTDS) system to generate arrangements to minimize the makespan of a task sequence with limited resources. The algorithm has two parts, which are synthetic cost construction and DRL processing to make arrangements. The synthetic cost represents the cost of different selections of arrangements in both resource usage and blockage arranging probability. This study uses the cost to measure the state-action value function to process the deep Q network (DQN) procedure to generate an optimized scheduling strategy. We establish the reinforcement learning strategy generation process by instantiating the computing components in the hardware as agents, and RAM resources and communication I/O ports as environment. A hardware-design-based decision rule is constructed to ensure that the computing variables are distributed as evenly as possible in storage, while making full use of the pipeline characteristics of FPGA. A compiler is written to generate an FRTDS binary stream to drive FRTDS. Accuracy and performance of the proposed method are verified and evaluated. We present simulation results of the modeling method, as well as from a classic method. Comparing these results, the makespan obtained by the proposed method is significantly shorter. It corresponds to the possibility of having higher computing power and dealing with larger-scale real-time simulation.

INDEX TERMS DQN, FPGA, power simulation, real-time, reinforcement learning, RTDS.

NOMENCLATURE

FRTDS	FPGA-based real-time digital simulator
DAG	directed acyclic graph
EST	earliest scheduled time
LST	latest scheduled time
f_i	current cost for dealing input
f_o	current cost for dealing output
f_{select}	current cost for control layer usage
f_{guess_1}	cost with arranging time exceed LST
f_{guess_2}	potential cost with subsequent tasks
Session	input, output, control layer handling period for certain task
SC	session cost
PC	prediction cost

The associate editor coordinating the review of this manuscript and approving it for publication was Canbing Li.

I. INTRODUCTION

Real-time simulation is of great significance to control system design, hardware equipment testing, and staff training. For instance, a low-cost real time simulation system based on a digital signal processor (DSP) was built for educational purposes [1]. [2] emphasized the importance of the analysis of microgrids with a real time digital simulator (RTDS) and [3] built a co-simulation framework that can assess microgrids with hardware-in-the-loop testing approaches. However, with the rapidly increasing simulation scale and the common use of electronic devices, demand for computing may exceed the supply. To effectively expand the scale of simulation under limited computing ability, a non-iterable method to shorten the time step and finish tasks with multiple FPGAs was presented [4]. However, the method is not always precise on different device models, and multiple FPGA is not a preferable choice considering its cost. [5] built a mathematical model of a device to shorten the time step. It makes the model more versatile but can only deal with one speed, and hence, it cannot simulate a combined system with multiple time steps. A dynamic averaging model was proposed to maintain precision with short time steps [6], but the method is not applicable if the focus is on the transient process and its control design.

Except for the above approaches, the problem can be considered from the perspective of hardware computing concurrent task-scheduling, which refers to tasks executed on hardware in parallel but not in parallel threads while scheduling. The task-scheduling process of cloud computing was optimized and computing efficiency improved by mathematically describing the load-balanced state [7]. A heuristic algorithm to solve multi-core task-scheduling problems under limited resources was presented [8]. FRTDS is also a multi-core system with limited resources, in which case the problem is transformed into a model that minimizes the makespan of concurrent task sequences with limited resources. For task-scheduling implementation, a solution to quantify the optimal substructure problem was introduced [9], but it can only generate results under sequential decision-making conditions. Resources were used as constraints and an artificial immune algorithm was used to plan and optimize task-scheduling strategies [10]. List and pack models for parallel task-scheduling were used while considering resource consumption, and an easily-deployed ε-approximation algorithm was presented [11]. This method balances algorithmic complexity and fast implementation, and cannot provide an optimal solution in theory. Load balancing and resource cost were quantified in a fitting function as optimized goals, and a PSO algorithm was used for task scheduling, but the heuristic method is complicated for expressions of different states, hence it is not easily used for swift migration [12]. Hardware resources were assumed to be unlimited, and the ideal scheduling time of each task was the optimal target benchmark. By making the actual scheduling time as close to the ideal as possible, the completion time of the computing task was shortened. This method only considers current limitations and not long-term impact [20]. The above methods take only resources as constraints and ignore the use of resources as decision-making variables.

This paper assumes that the proper use of resources can have a great effect on the task-scheduling process based on experience with imbalanced storage of variables in computing. We propose a reinforcement learning (RL)-based algorithm that takes resource usage as parameters to describe the cost of task selection, and whose principle is the balanced storage of variables as an arrangement. The resource usage of each hardware clock is described as a state, and the choice of a task to execute is a transition between states. Due to the non-full-state feedback of the state transition, there is a problem in describing resource usage under new states in later times. To describe the resource allocation caused by currently scheduled tasks in subsequent hardware use, a predicted window is established to estimate the subsequent impact of the current decision task. The predicted window obtains the future impact of the current task by analyzing its successor tasks in variable storage, and then we use these tasks to calculate a value based on its allocated resources. Also, to improve the balance of resource allocation, there is no fixed decision order of computing components, and we adopt load-balanced decision rules. In each hardware clock, all of the computing components must be sorted first and must make decisions with adequate communication.

The strength of this method is that implementation through deep reinforcement learning (DRL) enables it to find optimal sequential tasks in different types of equations and reorganize tasks in similar patterns without a training dataset or adjustments. Furthermore, the synthetic cost is based on resource usage, hence it can be applied to other systems that need to generate parallel part-sequential decisions with limited resources. When deployed, a shortcut based on hardware design calculates values as network outputs to accelerate the optimization. The feasibility of the proposed optimization method is demonstrated by two examples on an FRTDS system.

The remainder of this paper is organized as follows: Section 2 introduces FRTDS and the platform we use. Section 3 gives a clear view of the proposed method. Section 4 illustrates the implementation of an algorithm based on the proposed method. Section 5 explains our experimental results obtained with the proposed method and a comparative approach [20]. Section 6 discusses our conclusions and possible future work.

II. ABOUT FRTDS

FRTDS fulfills RTDS functions on an FPGA platform. FPGA-based RTDS is constructed in two ways. One is to use FPGA as a coprocessor [13]. The other [14] uses one or more FPGAs to compute in small time steps. Both ways rely greatly on component models to enable fully pipelined and parallel-computing characteristics. A frequency-dependent phase-domain (FDPD) line model was used to map a traveling-wave model to FPGA computing [13], and a switching network partitioning (SNP) model was used to map multiple converters to FPGA computing.

Our laboratory presents an FRTDS system, and we decompose power systems into subsystems to solve them. This is an effective approach [15], [17]–[21]. FRTDS casts the simulation process into many steps. The step size (50 μs) corresponds to a fixed number of 7500 hardware clocks. Considering frequency, we are working faster than [16]. One step-size length simulation is completed after solving all of the system equations. However, the number of hardware clocks corresponding to the step's completion time must be less than the maximum, which is 7500. When the simulation scale becomes larger, it takes more hardware clocks to solve the system equations in one step. With the same simulation object, compressing the number of hardware clocks representing the completion time improves the simulation efficiency, which means that the scheduling algorithms can handle larger scale simulation objects. To effectively reduce the data storage pressure of the FRTDS and improve the speed of simulation calculations, the multi-value parameters required by the direct calculation method (without solving the sub-network equations) were defined according to the sub-network in principle [17]. The hardware design and code flow generation of FRTDS were introduced [18]. GOOSE and SV communication interfaces were added to FRTDS, enabling its use for real-time simulation of intelligent substations [19]. Power system electromagnetic transient simulation calculation task scheduling software was developed to directly generate the binary stream required by FRTDS, eliminating concern about the coding of simulation programs [20]. The hardware platform we used to run examples was designed by [21], and it promises a reduction in LUT usage in FPGA. Figures 1 and 3 show the system structure of a computing component in FRTDS. In Figure 1, the computing component is connected to the RAM storage area through the read–write control layer, so the address of the RAM area belongs to the resources directly used by the system.

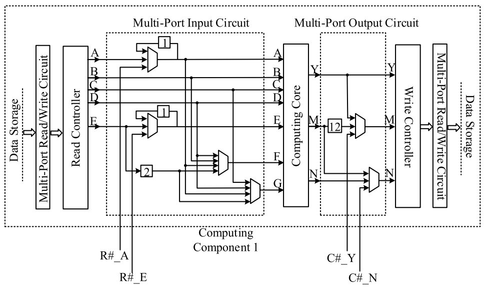


FIGURE 1. Structure of the computing component.

Regarding I/O ports, the task arrangement will directly interact with the RAM area. Though I/O ports belong to resources, they are mainly dependent on injection, which is determined by the task being scheduled. A task scheduling method refers to a computing expression corresponding to each input variable. The data are sent to the input I/O port, and the variable data are calculated according to the core structure to obtain the result from the output port. Therefore, there is at least one correspondence between a prescribed I/O port and an input variable in a computing task, that is, an I/O resource usage mode. A computing component can use I/O resources in accordance with the I/O port usage method specified by the computing task. It can use the F and G ports for direct data transmission after selection. It can also use the A or E port to connect to all RAM to obtain data directly. These two I/O usage methods enable the current computing component to communicate with other computing components. The use of ports F and G is based on selection. The real input data will only be input to the computing core through the five input ports A-E. If the I/O required by the computing task does not occupy all of the needed I/O ports, then the data are transferred from other RAM storage areas through the F or G port to complete the calculation. According to the hardware design [21], the A and E ports are directly connected to all RAM memory blocks, so data in any RAM area can be obtained directly by reading the control layer, whether in the private RAM area or that of the computing component. The reading and writing relation between RAMs and computing components can be seen in Figure 2. The double-ended arrow means RAM is private for certain components, the red arrow means it is not private but can be accessed by the control layer, and the black arrow means the output to RAM with control layer. As to the two data communication methods, the latter directly consumes the communication capacity by taking up I/O resources, and the former can retain data communication ability if I/O resources are available for computing components.

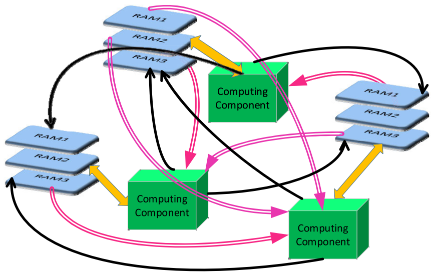


FIGURE 2. Relation between computing components and RAM areas.

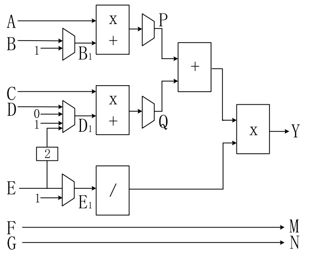


FIGURE 3. Structure of computing core.

III. MODELING METHOD BASED ON RESOURCES

A. BLOCKAGES AND COSTS

We construct cost to measure the interrupted working occasions of computing components. Assuming no RAM read and write restrictions, the deep pipeline characteristics of FPGA will be obtained and the hardware will always be at full capacity with the highest efficiency. This will result in a partial serial relationship between tasks, which will obtain the EST and ideal completion time of all computing tasks. The LST represents that no computing task will affect the ideal completion time through DAG dependency, and each can be obtained through reverse recursion. Under ideal conditions, no blocks occur and the computing components can fully work at each hardware clock to reach the optimal state. Due to limited resources, computing tasks cannot always be arranged before their LST. For the analysis of the operating modes of computing components, the variables used for computing may exist in their own private RAM area or in the RAM of other computing components. In the latter case, it is impossible for computing components to directly read variables. The hardware provides communication methods, but the I/O resources are limited, and cannot guarantee the smooth flow of reading and writing variables. A computing component may not always be in working condition. Define a block as a situation where certain computing components have communication conflicts or RAM read-and-write conflicts and cannot execute a computing task at a certain hardware clock. Through our previous experiments, we assume the most fundamental reason for a block is that the variables used for calculation cannot be reasonably and evenly distributed in the storage area during scheduling. The communication capabilities of the hardware can alleviate this imbalance, but the task scheduling algorithm fails to consider their use. Hence, there will still be over-communication-based scheduling schemes that eventually lead to blocks.

The solution is illustrated in Figure 4. The framework is in a closed-loop feedback state. Resources influence the strategies of distribution by arranging tasks, and distribution influences resource usage thorough agents (computing components). Our solution to the task scheduling problem under conditions of limited resources is to consider the resource usage in the scheduling algorithm and fully guarantee the reasonable storage allocation of the computing variables while scheduling. Calculation tasks use different variables and cause different potential blocking situations. We use cost to measure the effect of potential blocks. The cost concept corresponds to the introduction of blocks when using different modes of tasks for scheduling. Based on different effects on resources, we divide the cost function into an SC part that characterizes the current resource occupancy and a PC part that evaluates the current decision task's future effect. Hence, the method in this paper considers the synthetic cost for each task arrangement, considering the impact on current resource occupation and predicting possible future blockages. We use SC and PC to design the computing component selection criteria and the principles controlling the allocation of computing variables in the storage area.

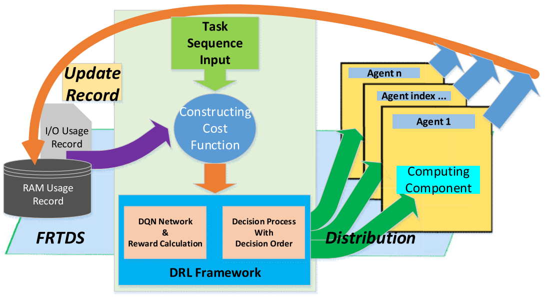


FIGURE 4. Proposed solution framework.

B. CONSTRUCTION OF COST FUNCTION FOR SINGLE-COMPUTING COMPONENT

1) SESSION COST CALCULATION

FRTDS can carry multiple computing components, which can independently execute computing tasks. It can also perform data communication based on communication capabilities to help other computing components complete calculations. We first analyze SC in a single-computing component. To facilitate the description of resource usage, we label the resources of FRTDS. As seen in Figure 1, multiple selected read-write control I/O ports are connected to all RAM in the RAM area, and the remaining I/O ports only communicate with the private RAM area of their own computing components. According to the design of FRTDS [21], internal RAM can store up to 1024 double floating-point numbers. Let FRTDS enable L+1 BRAMs labeled n_0 to n_L, and let the system contains P+1 computing component modules labeled m_0 to m_P. At the same time, we use $qi \in [1, 5]$ as the input port on each computing component. Similarly, the output ports are qo and $qo \in [1, 3]$. Finally, we use ADR to represent the constant 1024. To better express the many-to-many mapping relationship between I/O ports and RAM, we use adr_{qi} to express the correspondence between the I/O port used by the RAM of variables in the currently considered task, and we express a relationship representing a many-to-many connection as a one-to-one connection when scheduling. According to the actual computing process, a session can be divided into input, selection control, and output processes. We express f_i, a part of SC, from the input area of the computing component during task scheduling as follows:

$$fi = \sum_{qi \in I} \frac{\left(c_1 \cdot adr_{qi}\right) pi}{ADR} + \sum_{qi \in I} \frac{c_2 \cdot (ADR - adr_{qi}) pi}{ADR}, \quad (1)$$

where adr_{qi} is the number of addresses that have been used in the currently mapped RAM, and I is the set of RAMs corresponded to input variables. P_{qi} takes a discrete value to indicate that the RAM occupies several working ports when used. If only one is occupied, it is set to 1. Taking up both ports for RAM blocks the use of other variables in the same RAM. This will lead to imbalance of both arrangements and loads. We double the cost so as to avoid such occasions. c_1 and c_2 are hyperparameters to adjust the weights of the model to measure costs. Similarly, f_o can be expressed as follows:

$$fo = \sum_{qo \in O} \left(\frac{c_3 \cdot adr_{qo} + c_4 \cdot (ADR - adr_{qo})}{ADR}\right) \cdot P_{qo}, \quad (2)$$

which considers that the used address is blocked and the free address cannot be used. We should make clear that qi and qo represent input and output I/O indices and O is the set of RAMs corresponding to outputs.

A session includes a control layer that can complete the communication between the current computing component and others. According to the two communication methods, the cost of communication is different for the two methods. The use of global RAM reading requires only the RAM address because the remaining communication capacity is retained, while the F or G port for pipe communication requires communication capacity, which will communicate when the remaining computing components are making schedules. We add additional f_{select} in this situation. Since communication can be divided into the modes of obtaining and providing data, we write (3), as shown at the bottom of this page, where I_{select_o} is the set of M and N ports of any computing component, I_{select_i} is the set of F and G ports of any computing component, and $I_{A/E}$ is the set of A and E ports of any computing component. P_o equals 1 if the o^{th} selection controls when the I/O output port is used for pipe communication, and is zero otherwise. P_i and P_k are defined analogously for the input of F and G ports and A and E ports, respectively. $P_{F/G}$ indicates the way to use the F or G port to communicate with other computing components. From the computing kernel structure in Figure 2, we can see that M must be used as the output when using the F port, so we set $P_{F/G}$ to 1 when pipeline communication between F and G is not applicable or only one of F and G is used. When the F and G ports are used at the same time, $P_{F/G}$ is set to 2; N represents the number of all computing components. The index is the decision-making order of each component at each moment, and the order is a number obtained by component sorting results. Equation (3), includes the cost of communication to obtain data, data provided by communication, and non-pipeline communication using any A or E port to read any RAM. The first part of the equation represents the data acquisition and interactive blocking cost, the second represents the blocking cost of providing data for interaction, and the third is the blocking cost of global data access. This paper considers the importance of communication capabilities for reasonable task scheduling of other computing components in the future, and increases the SC of computing components that are scheduled for task execution and communication. Based on the above analysis, the SC of a single computing component can be written as follows:

$$SC = fi + fo + f_{select}. \tag{4}$$

2) PC WINDOW AND PC CALCULATION

According to the previous analysis, blocks emerge when conflicts exist in reading and writing addresses or communicating, because the full-state feedback of resources at a certain hardware clock in the future cannot be explicitly obtained. The cost of potential blocks cannot be directly described by address occupation. We propose the concept of a PC window from another perspective to describe possible blockages caused by task scheduling over a period.

The PC window is the closed interval formed between the current decision task time and the ideal lasted arranging time of the succeeding task being tested, which is jointly determined by all subsequent tasks of the current task and time. The length of the window limits the current decision task to consider each subsequent task in turn. The arrangement of these tasks can be described by predictable computing and communication needs. The window must be changed according to each successor task. The prediction method proposed in this paper requires a window analysis and summation of all successor variables. The concept of the window can be understood from Figure 5. Task y is the successor task of tasks x1, x2, and x3 and the previous task of tasks z1, z2, and z3, according to DAG dependency. Taking task y as an example, each successor task establishes a PC window, and the union of all PC windows of successor tasks constitutes the PC window of task y.

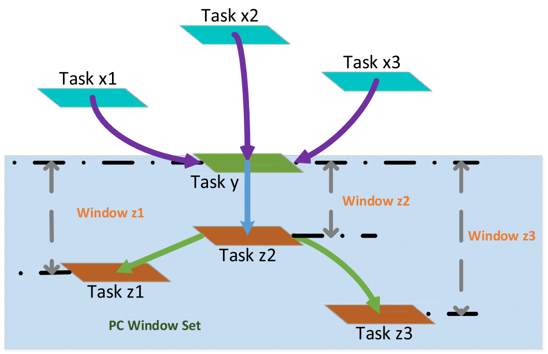


FIGURE 5. Concept of predicted window.

Let the current decision task have j successor tasks; i represents the i^{th} calculated variable in the j^{th} successor task, and variable(j,i) represents whether the i^{th} variable of the j^{th} successor task is uncalculated (variable not in the RAM storage area in the time between the predecessor and successor tasks in the PC window), and if so is set to 1, and otherwise is 0; cls represents the ideal scheduling time of the successor task, cur represents the current decision time, and lst represents the LST for subsequent tasks. The blocking risk introduced by uncalculated variables in the PC calculation is as follows:

$$f_{guess\_1} = \begin{cases} \sum_j \dfrac{\sum \text{variable}(j,i)}{(cls - cur)\cdot(lst - cur)}, & \text{cur} < \text{cls} \\ \sum_j \dfrac{\sum \text{variable}(j,i)}{(cur - cls)\cdot(lst - cur)}, & \text{cur} > \text{cls} \\ 0, & cur = cls. \end{cases} \tag{5}$$

$$f_{select} = \left\{ \begin{aligned} &\sum_{i\in I_{select\_i}} \left(\frac{c_5 \times adr_{Ii} + c_6 \times (ADR - adr_{Ii})}{ADR} + \frac{1}{2}\right)\cdot P_i \cdot P_{F/G} \\ &+ \sum_{o\in I_{select\_o}} \left(\left(\frac{c_3 \times adr_{Io} + c_4 \times (ADR - adr_{Io})}{ADR} + \frac{1}{2}\right)\cdot P_o\right) \times (2 - P_{F/G}) \end{aligned} \right\} \times (N - index)$$

$$+ \sum_{k\in I_{A/E}} \frac{c_7 \times adr_{Ik} + c_8 \times (ADR - adr_{Ik})}{ADR}\cdot P_k \tag{3}$$

This part of the PC is described by the relationship of task scheduling times. A task that can be scheduled at the ideal scheduling time is considered non-blocking. Otherwise, the result will be added and the blocking cost will change the sign when the time has exceeded the LST, which means that f_{guess_1} encourages the task to complete the scheduling as soon as possible and reduces PC by completing the arrangement before the LST arrives to make it complete as soon as possible. Based on the common sense of the scheduling process, PC needs to involve the estimation of the communication cost. Even if a variable exists in the RAM storage area before the specified time, there are still cases when the RAM does not belong to the private RAM of the current computing component. In this case, the current task selection will need communication to arrange subsequent tasks, which will incur additional blocking costs. We calculate this part of PC as follows:

$$f_{guess\_2} = \sum_{j \in J} \frac{N(j)_{ready} - 1}{\text{ready}(j) + 1}, \tag{6}$$

where j represents the j^{th} successor task and J is the set of all successor tasks for a considered task, and ready() counts the ready variables of the j^{th} successor task that must be fetched from other components' RAM areas when used in the j^{th} successor task. $N(j)_{ready}$ indicates the number of variables that are ready in the j^{th} successor task. Based on the above analysis, the PC of a task can be expressed as follows:

$$PC = f_{guess\_1} + f_{guess\_2}. \tag{7}$$

And synthetic cost for certain task we use in the following passage is the sum of SC and PC for certain task.

IV. DRL-BASED OPTIMIZED TASK DECISION

A. INTRODUCTION TO MULTI-AGENT DRL

Reinforcement learning determines the optimal behavior of an object by interacting with the environment based on its state. Unlike supervised learning methods, reinforcement learning requires no clearly labeled training set. Optimization is accomplished by continuous interaction with the environment to obtain feedback and modify object selection strategies. Reinforcement learning is unique because the feedback it gets from interacting with the environment is time-delayed, which enables it to adapt to complex decision-making problems. Its interaction model is shown in Figure 6.

The environment varies continuously with the action of the object between states. The environment is modeled from the perspective of the Markov decision process. Most such methods are carried out through deep learning networks. There are several construction methods for deep reinforcement learning, among which DQN networks are commonly used. A DQN network is based on Q-learning, where Q stands for the state-action value function, which is used to measure the value obtained by selecting a certain behavior a with current state s in the current object. To select the research object (agent) is to select the action by using the existing

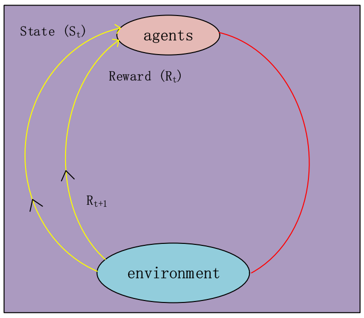


FIGURE 6. Concept of DRL interaction.

state-action pair Q value as an estimated value, and the valuation network to calculate the actual state-action selection behavior of Q and update it after the selection. To continuously make selections can continuously improve the effect of the agent's next selection. The Q value calculation of the multi-agent system is introduced below.

The learning goal is defined as learning strategy π: S $\in$ A, the finite state set S = {Si} and A = {ai} as the action set of the object. The strategy description object selects the probability distribution {P1, P2,...Pi} of action $a \in$ A according to the current state $s \in$ S. Starting from any state S_t, the expected cumulative attenuation obtained according to strategy π^* is as follows:

$$V^{\pi}(s_t) = E\left(\sum\nolimits_{i=0}^{\infty} \gamma^i r_{t+i}\right), \tag{8}$$

where $0 \leq \gamma \leq 1$ is the delayed reward attenuation coefficient, which reflects the importance of the current and future reward in the total reward, and r_t is the bounded return obtained from each hardware clock. Considering that the system state transition is a nondeterministic Markov process, the expectation is added to the calculation of cumulative returns. The best strategy of the object is to maximize (8).

The behavior selection can be simplified to the evaluation process of the Q-estimation network, and network estimates are calculated according to (9). For convenience, we use $\hat{Q}$ to denote the Q-estimation value, and Q is used to calculate the Q value for behavior selection, which is calculated through the delayed reward principle:

$$\begin{aligned} Q(s, \vec{a}) &= E[r(s, \vec{a}) + \gamma V^*(\delta(s, \vec{a}))] \\ &= E(r(s, \vec{a})) + \gamma E[V^*(\delta(s, \vec{a}))] \\ &= E(r(s, \vec{a})) + \gamma \sum_{s'} P(s'|s, \vec{a}) \cdot V^*(s'). \end{aligned} \tag{9}$$

Equation (9) is iterative. This optimization problem is considered from the perspective of dynamic programming of optimal substructures. We replace (9) with

$$Q(s, \vec{a}) = E(r(s, \vec{a})) + \gamma \sum_{s'} P(s'|s, \vec{a}) \cdot \max_{\vec{a}'} \hat{Q}(s', \vec{a}'), \tag{10}$$

where $r(s, \vec{a})$ is the reward that a selection can bring at this moment; $\delta(s, \vec{a})$ is the state successor function that represents the next entry of the object after the component acts in state s with action a; $P(s'|s, \vec{a})$ is the probability that the subsequent state is s' after multiple objects take the action vector in state s, and a' is the action of each research object in the new state s'. In theory, iteration should stop when the agent transfers to the final state. Besides, since taking max as the target estimation in (10) may cause the estimated value to be significantly higher than the real value because of using same set of parameters in networks. There is also a double DQN method for balanced value estimation. This method introduces an additional set of DQN networks as a target-DQN with different parameters, which is designed for Q value calculation; the main DQN is used for behavior selection and the target-DQN Q value converges to the main DQN to ensure that the training process does not oscillate. Equation (10) can be modified to:

$$Q(s, \vec{a}) = E(r(s, \vec{a})) + \gamma \sum_{s'} P(s'|s, \vec{a})\hat{Q}(s', \arg\max_{\vec{a}'} Q(s', \vec{a}')), \quad (11)$$

where Q is the main DQN and $\hat{Q}$ is the target-DQN. The Q value for behavior selection is provided by the main DQN, and then the Q estimate value can be calculated using the target-DQN. Parameters of target-DQN are often set as parameters of the main DQN after several steps. The Q value of the main DQN can be split into two parts for calculation, which will increase the speed of the agent's behavior selection in the main DQN. According to the typical method, we obtain the Q value of the main DQN network as follows:

$$Q(s, \vec{a}) = V(s) + \sum_{\vec{a}} A(s, \vec{a}), \quad (12)$$

where $V(s)$ estimates the current state value and *A(s,a)* estimates the advantages of the relevant actions.

If the network parameters are integrated into the gain function and state successor functions as parameters and are expressed as $r_t(s, \vec{a}; \theta_t)$ and $\delta_t(s, \vec{a}; \theta_t)$, respectively, then the network parameters can be updated according to the gradient based method:

$$\theta_{t+1} = \theta_t + \varepsilon \cdot (Q(s_t, \vec{a_t}; \theta_t) - \hat{Q}(s_t, \vec{a_t}; \theta_t)) \cdot \nabla_{\theta_t}\hat{Q}(s_t, \vec{a_t}; \theta_t), \quad (13)$$

where ε is the learning rate of the neural network, and the remaining quantities are calculated by Eqs. (11) and (12). We generally set γ between 0.97 and 0.99. Q estimation can be updated by:

$$\hat{Q}_{t+1}(s, \vec{a}) = \hat{Q}_t(s, \vec{a}) + \beta_t\{r_t(s, \vec{a}) + \gamma \sum \lambda_1^* \prod_{i=2}^{n} \lambda_{it}\hat{Q}_t(s', \vec{a}')\}, \quad (14)$$

where λ_1^* is the estimated probability of the current agent executing the optimal action strategy based on its judgment, and λ_i is the estimated probability of the current agent executing strategy i that it is currently capable of executing. Probability estimates can be calculated using roulette or softmax methods. Equations (13) and (14) are used in network training with stored experiences in (state, reward, action, next state) format. In theory, we stop the iteration when our agent transfers to the final state. However, in practice, when training the DQN network, we often use:

$$loss = \frac{1}{2}\sum_{a_i \in \vec{a}} \left(r + \gamma\hat{Q}\left(s, \arg\max Q\left(s, a_i; \theta^-\right); \theta'\right) - Q\left(s, a_i; \theta^-\right)\right)^2 \quad (15)$$

where r is the reward of the current action, a_i is the action chosen, and θ is the set of network parameters. The equation represents the summed square error of the target-DQN Q value and main DQN Q value when choosing certain action vectors. We use this loss for backpropagation in (13), and update the Q value by a time discrete update:

$$Q\left(s, a_i\right) = Q\left(s, a_i\right) + \alpha\left(r + \gamma\hat{Q}\left(s, \arg\max Q\left(s, a_i; \theta^-\right); \theta'\right) - Q\left(s, a_i; \theta^-\right)\right) \quad (16)$$

When the summed square error is within tolerance, we stop the iteration. In our implementation, we add a max epochs limit to stop the iteration in the case of exceptions.

B. REINFORCEMENT LEARNING BASED ON COST

To develop DRL, we must set the agent and environment, choice of strategy, and expression of state values. The FRTDS system has multiple computing components, each needing to make task scheduling decisions, which will affect the allocation of RAM storage resources. It is obvious that FRTDS and multi-agent DRL bear a resemblance. We abstract each computing component in the FRTDS system as an agent, and we abstract task selection as an action in DRL. Since we have a fixed number of task modes, we have a finite action space. We quantify resources in state expression by making resource variables as input vectors. We update the RAM port usage and address usage percentage according to task selection for each hardware clock. RAM port usage will decide whether certain RAM can be used at certain hardware clocks, and we encode this as 1 if it can, and zero otherwise. We add numbers of undone tasks to it. We extend the address-use percentage of each component to the index we have just encoded. We map a small range of values to one figure which is the mean of this range. We suggest a $(-2\%, +2\%)$ range. Now we can make continues filed into discrete field. We clip the value mainly to shrink the scale of the state space to reduce the computing complexity. Then we will have an input vector representing a certain state. We explain how to combine SC and PC with the FRTDS system by mathematical principles of reinforcement learning to better generate strategies. With a finite action and state space, we now can implement our algorithm on the DQN structure.

The state transition process based on the use of resources is different from the general state transition process. FPGA has deep pipeline characteristics, so the calculation task selection vector executed in the current state will take effect sometime later, depending on the length of the pipeline [21]. The state transition with pipeline characteristics will affect the calculation of conditional probability in (11) For subsequent states in conditional probability calculation, we propose to use the state after the pipeline length instead of that at the next moment on the hardware clock, hence the state in the equation is the performance after the influence of the execution of selected calculation tasks. This gives the calculation practical meaning. And indeed, this system should be seen as history-dependent.

For $V(s)$, as the system's current state value estimate, it can be obtained by the main DQN network output added with a task-considering bias. The main DQN network takes input vectors we build and generates evaluated state values as outputs. We use LST to describe the bias with task arrangement status:

$$\text{bias}(s, \vec{a}) = \sum_{i\in\vec{a}} (lst + 100 - t_i)^2, \tag{17}$$

where t_i is the time schedule of the task scheduled by the i^{th} computing component, and lst is the LST of the task scheduled by the i^{th} computing component, where 100 is the hardware clock length, which can be adjusted according to the hardware design. For the reward system, with limited resources, we still must ensure that the task accepts a certain reward when finished. Therefore, considering the longest pipeline length, we set the buffer value to 100. The threshold time is LST plus the buffer value. Negative rewards will be received when arranging tasks after their threshold time has passed, and positive rewards will be received otherwise.

For the calculation of $A(s, a)$, extended summation can be combined with the SC and PC described:

$$A(s, \vec{a}) = -\sum\nolimits_{i=0}^{P} (SC(m_i) + PC(m_i)), \tag{18}$$

where m_i is the task index chosen by computing component m_i. The set of all m_i task scheduling choices constitutes the computing task selection vector a.

C. DECISION RULES AND DECISION ORDER OF COMPUTING COMPONENTS

The decision rule is to optimize the target service so as to better balance the computing variables of the task scheduling process. It is a basic decision rule formulated to eliminate blocking. The SC mentioned above can only be used as part of the decision rule. Eliminating blocking in the task scheduling process can more efficiently schedule computing tasks, which is the starting point of our optimization work.

Based on whether the variables used for the operation have been stored in the RAM area and the dependency constraints of the calculation task, we screen out a ready task sequence that can be considered for each operation component at each moment. To maintain the reasonable allocation of variables in the storage area during the execution of computing tasks,. We propose the following decision rules:

1. According to the obtained hardware resource usage description and ready limit, all of the ready task sequence is generated before the beginning of a decision time.

2. We check whether all ready tasks can use the hardware read-write control layer for the computing component currently making a decision.

3. We check whether ready tasks share variables. If so, then we establish group scheduling using control-layer design to broadcast variables. Otherwise, we estimate the synthetic cost of a task according to its SC and PC and obtain the cost-sorted sequence by filtering the ready task sequence.

4. We use the ε-greedy method to randomly select all currently selectable tasks by generating a random number. If the number is less than ε, then the current task is selected, and otherwise, tasks should be chosen by the greedy rule with the least synthetic cost.

5. After obtaining the task, we determine how many variables must be obtained through communication. If there is more than one, the calculation task will be locally optimized and the variables will be calculated in advance according to the resource usage record through the time when there is an idle pipeline communication capability. For output, we use the PC window to analyze the subsequent tasks of the output variable again and allocate the output to the most concentrated RAM according to the storage allocation of each variable of subsequent tasks.

6. In this process, the remaining variables may not be ready. At this time, to ensure a balanced load, the RAM with the greatest number of unused addresses for output should be chosen. Also, the remaining variables may be evenly distributed with multiple RAMs. At this time, multiple allocations of the output port should be used to allocate to multiple RAMs. This can minimize the imbalanced resource allocation of computing components.

Computing components are designed so as to not make decisions in fixed order at each hardware clock. The decision order should be dynamically changed, which affects the occupation of RAM and communication resources. We propose the following rule to determine the decision order of the computing components at each hardware clock. We consider the goal of load balancing and sort them in descending order by calculating the address space that each computing component has already occupied, and expand the operation arrangement according to this order.

This order is used because computing components occupying a large number of addresses have a higher probability to execute computing tasks without communication, so communication capabilities are reserved for subsequent tasks, which can improve scheduling efficiency. Using the above decision rules and order, the resource and synthetic cost description of FRTDS can be used to complete the generation of the calculation task selection vector a.

D. COMPUTING TASK OPTIMIZED SCHEDULING ALGORITHM BASED ON FRTDS

The algorithm in this paper is designed to improve the rationality and balance of variable allocation in storage during the scheduling of computing tasks. At the same time, we assume that the allocation of initial variables affects the overall scheduling. The method of initial parameter allocation is to set variables in RAMs according to the actual topology of the system to be simulated, for example, to allocate data on the same bus line to the same computing component's RAM storage. The logic of the algorithm in this paper at each moment can be summarized as follows.

1. Update hardware resource usage.
2. Determine which computing component should be assigned a computing task according to the decision order.
3. Filter the ready task sequence according to the ready condition and calculate the SC in conjunction with the computing component.
4. Select the calculation tasks to be performed for the current computing component according to the decision rules.
5. Set the current computing component that has been assigned a computing task and skip to step 1.
6. Loop until all computing components at the current moment get a computing task or no ready task exists.
7. If all computing components have accepted a task assignment, or the current task assignment is complete, then calculate the current and delayed rewards.
8. End the current time; add 1 to the global time record.

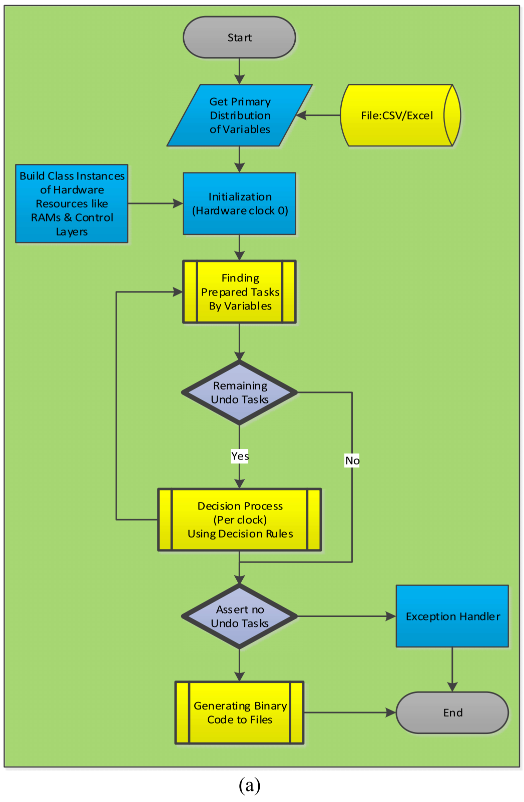
The algorithm can be seen in Figure 7a, and the decision loop in Figure 7b.

To illustrate the details, the logic of implementation can be seen in Algorithms 1 and 2. Algorithm 1 shows how the strategies are generated. *FormulaCompile* is used to read in tasks and build DAG dependency. *DelayRewardUpdate* accumulates long-term rewards in the DRL environment. *BinaryCodeGen* generates a binary stream that FPGA can use.

Algorithm 2 shows the execution of the Q-learning process to explore solutions. DQN network is included in *CostFunction*.

V. FRTDS EXAMPLE VERIFICATION AND RESULTS ANALYSIS

We first implement instruction compiling software by using the concept of coding flow [18], and extend the software's computational task scheduling algorithm based on synthetic cost and reinforcement learning. We use Qinbei 220 kV and 500 kV power plant real-time simulation calculation scripts to compare the proposed algorithm with FRTDS. The topology of the platforms can be seen in Figure 8a and b. We first verify the correctness of the results using the proposed algorithm. The real-time simulation result is compared with the PSCAD offline simulation result. We analyze results we get under the same script and hardware resources using the two methods. According to Qinbei 220kV power plant simulation, we select the PT voltage value of index 221 as the observation object, set the BC two-phase short-circuit fault in 4 s, and clear the fault after 0.2 s.


Start
Get Primary Distribution of Variables
File:CSV/Excel
Build Class Instances of Hardware Resources like RAMs & Control Layers
Initialization (Hardware clock 0)
Finding Prepared Tasks By Variables
Remaining Undo Tasks
Yes
No
Decision Process (Per clock) Using Decision Rules
Assert no Undo Tasks
Exception Handler
Generating Binary Code to Files
End
(a)

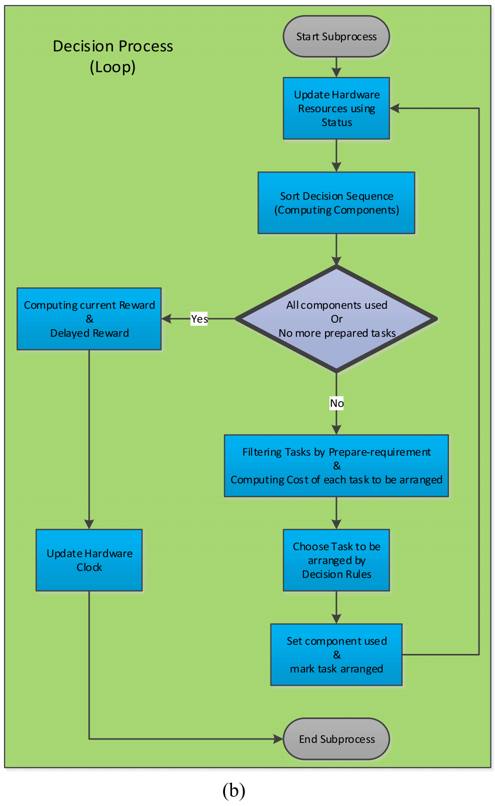

Decision Process (Loop)
Start Subprocess
Update Hardware Resources using Status
Sort Decision Sequence (Computing Components)
All components used Or No more prepared tasks
Yes
Computing current Reward & Delayed Reward
No
Filtering Tasks by Prepare-requirement & Computing Cost of each task to be arranged
Update Hardware Clock
Choose Task to be arranged by Decision Rules
Set component used & mark task arranged
End Subprocess
(b)

FIGURE 7. (a) Process of proposed algorithm. (b) Decision loop.

Algorithm 1 Task Scheduling

```
task_sequence, dag ← FormulaCompile(input, calc_rules)
undo_tasks ← len(task_sequence), Q_table ← { }
env.reset(), clock ← 0, done_tasks ← { }, delay_reward ← 0
strategy ← { }
while undo_tasks > 0 do
    for each task in task_sequence do
        if task.dag_limit==false && task.can_arrange then
            can_do.append(task)
        end if
    end for
    state ← env.update()
    do_tasks ← QExplore(state, env, can_do, Q_table)
    obs_timestamp ← env.step(do_tasks, strategy)
    DelayRewardUpdate(delay_reward, obs_timestamp)
    undo_tasks ← undo_tasks – len(do_tasks)
    done_tasks.extend(do_tasks)
    task_sequence ← task_sequence \ done_tasks
    clock ← clock + 1
end while
BinaryCodeGen(strategy)
returnstrategy
```

Algorithm 2 *QExplore(state, env, can_do, Q_table)*

```
deo ← DecisionRuleSort(state.component, env)
while any state.component is available or len(can_do)>0
 with deo as decision_order do
  min_cost ← inf
  for each task in can_do do
    if(env.resource, task.mode) in Q_table.keys() do
      cost ← $\alpha Q+(1-\alpha)$CostFunction(env.resource, task)
    else
      cost ← CostFunction(env.resource, task)
    end if
    (cost < min_cost)?(min_cost=cost):(min_cost)
  end for
  state.op_task.append(ct ← FindtaskbyCost(min_cost))
  can_do.remove(state.op_task)
  Q_table.update()
  state.ramuse.record(rec ← RAMArrange (ct, state, env))
end while
return do_tasks ← doTaskfactory(state, env, Q_table)
```

Taking 220 kV as an example, the change of the PT voltage is shown in Figure 9, where the solid line is the result obtained by our algorithm and the dotted line is the offline simulation result of PSCAD. We can see from Figure 9 that the two curves mostly fit. Then we use Qinbei 220 kV and 500 kV power plant scripts to execute the two algorithms to compute task scheduling. The makespan of the two algorithms on the two scripts are shown in Table 1.

In the scheduling process of the two algorithms, the time distribution of the RAM read and write code corresponding to each computing component (three in total) with 220 kV and 500 kV simulations are shown in Figures 10a, b and 11a, b, where code time represents the concept of the hardware clock mentioned above, and the instruction refers to the 16-bit binary code in each component. We set the number of codes for each component as the y-axis, and the hardware clock as the x-axis, and then we plot the curves. The colored zone represents that the number of codes varies from the lower to upper bound, and the lower bound could be zero. The number of codes remains zero for a period will leave a blank on x-axis which means certain component runs into block. The statistical results of the communication times used by the two algorithms in the scheduling process and the total number of codes are shown in Table 2. We set the initial variables only in computing component 1 (as PE1) in the 500 kV simulation. This is different from balanced storage in the 220 kV simulation.

Real-time simulation solves system equations in one step and solves them again in the next step. However, the computing process is the same. So, we record the statistics in just one step for our comparative analysis.

TABLE 1. Makespan comparison.

Simulation Object	Proposed Algo Makespan (hardware clock)	[20] Algo Makespan (hardware clock)
220 kV	3527	5648
500 kV	5321	6839

TABLE 2. Comparison of code usage count.

Simulation Object	Proposed Algo (Communication count)	Algo in [20] (Communication count)	Proposed Algo (Total Code count)	Algo in [20] (Total Code count)
220 kV	9729	26015	37353	85801
500 kV	16458	48385	87147	104897

We can make the following basic observations from the graphs and tables.

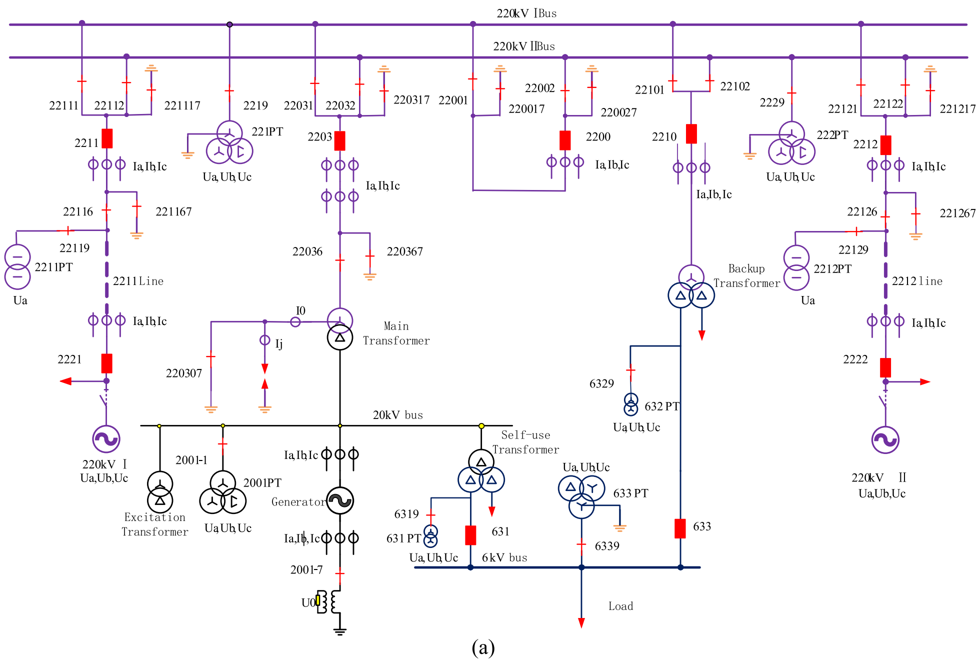
1. The makespan (in the hardware clock) obtained by the proposed algorithm is significantly shorter than from the method in [20], considering Tables 1 and 5.

2. In Figure 10a, there is no blank on the x-axis (interruption of instructions) in one simulation step.

3. In Figure 10b, obvious interruptions of instructions occur at about 4200 hardware clocks.

4. The range between the upper and lower bound is wider in Figures 10b, 11b than 10a and 11a.

5. There is still a blank on the x-axis when using the proposed method in Figure 11a with the second component.



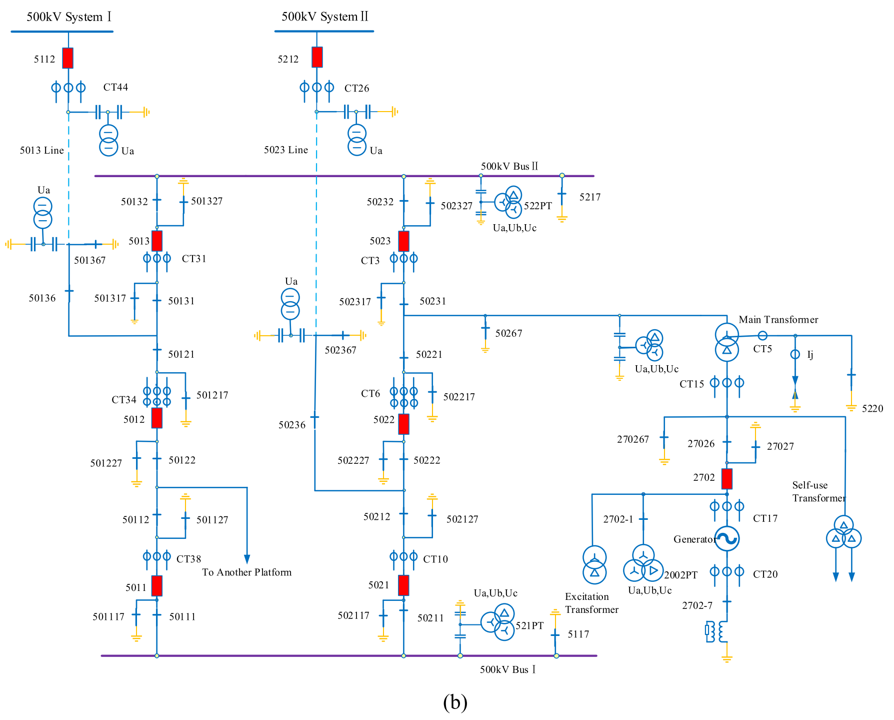


FIGURE 8. (a) 220kV power plant. (b) 500kV power plant.

6. Communication approaches' using percentage of the proposed method is far lower than [20] when considering Table 2.

7. Both methods basically achieve load-balancing according to Figures 12 and 13.

Considering the solving methods of the system equations, this paper indicates that the interruption phenomenon is due to the method of solving system equations. Taking the Gauss method as an example, it is likely that there is a serial block due to DAG dependency when there are no conditions for parallel computing. The same issue can be found in the second component in Figure 11a near 3700 on the x-axis. We believe that this blank can be filled by changing the method to solve the system. Comparing the fluctuations of Figure 10a and b and the number of communications counted in Table 2, it can be found that the fluctuation in Figure 10a is

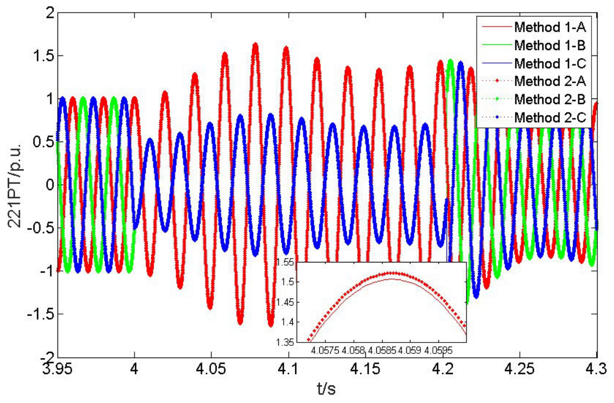


FIGURE 9. Results compared with PSCAD.

TABLE 3. load of each component record/220 kV.

Statistics	Proposed	[20]
PE1	19334	31695
PE2	16784	28093
PE3	16335	26013
Total	37353	85801

TABLE 4. Load of each component record/500 kV.

Statistics	Proposed	[20]
PE1	39821	41767
PE2	23440	32381
PE3	23886	30749
Total	87147	104897

more stable (narrow range between lower and upper bound). It can be found from Table 2 that the communication using a percentage of the proposed method is far lower than that for [20], which we think proves that a higher percentage of communication leads to a longer makespan. This is also evidence that balancing the allocation of variables can have a better effect in this process. Comparing Figures 10 and 11 with the same method, we can tell that the trend of graphs is mainly dependent on the Gaussian solution method. The growth of makespan does not vary with the total number of codes in a linear relation. This to some extent means larger scale simulation objects still can be compressed in makespan compared with the fixed number of 7500.

Through the above results, we conclude that our proposed method has the effect of keeping working loads in balance, but the effect is not always good with the influence of the initial distribution of variables. However, overall, the method tries to keep a balance through balanced allocation, and it is better than the classic method [20] when the initial storage of variables is set.

We use some other simulation instances to further prove that the proposed method shows better performance in handling different simulation objects. The comparative results can be seen in Table 5, which [1], [2] represent for different solving methods of system equations. Xi'an, Guangzhou stand for different simulation objects.

Since the decision rule proposed by the algorithm in this paper explicitly considers the balanced allocation of

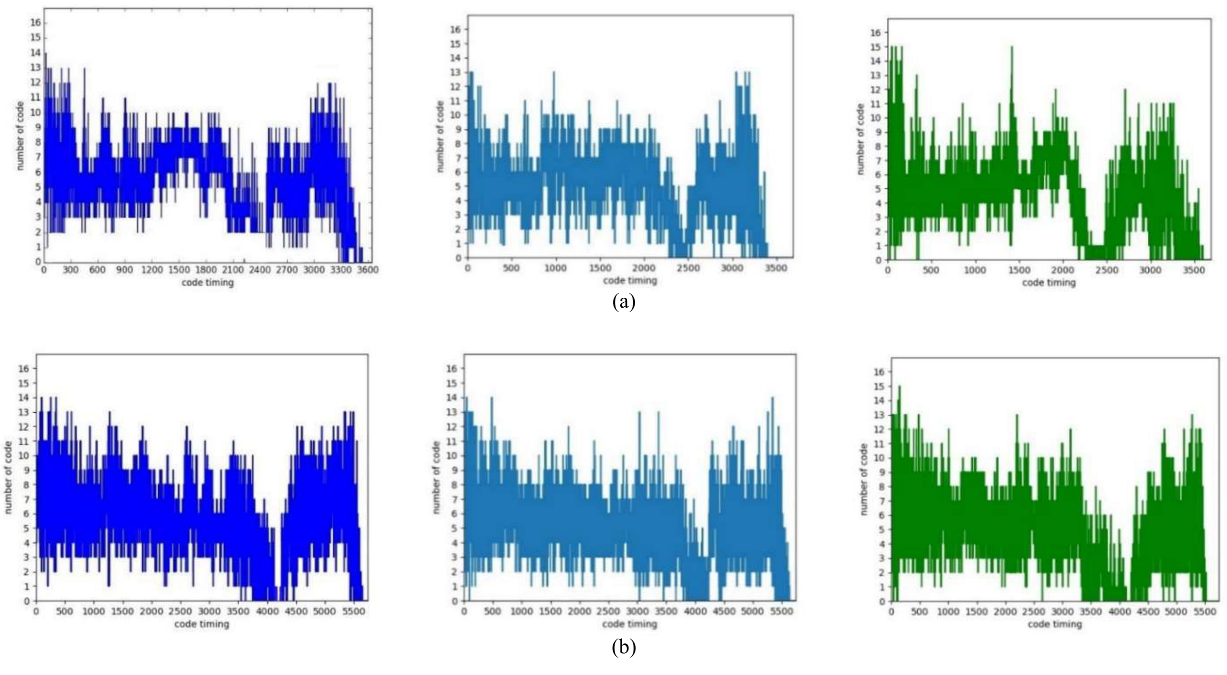


FIGURE 10. (a) Result of the proposed method with each PE/220kV. (b) Result of [20] with each PE/220kV.

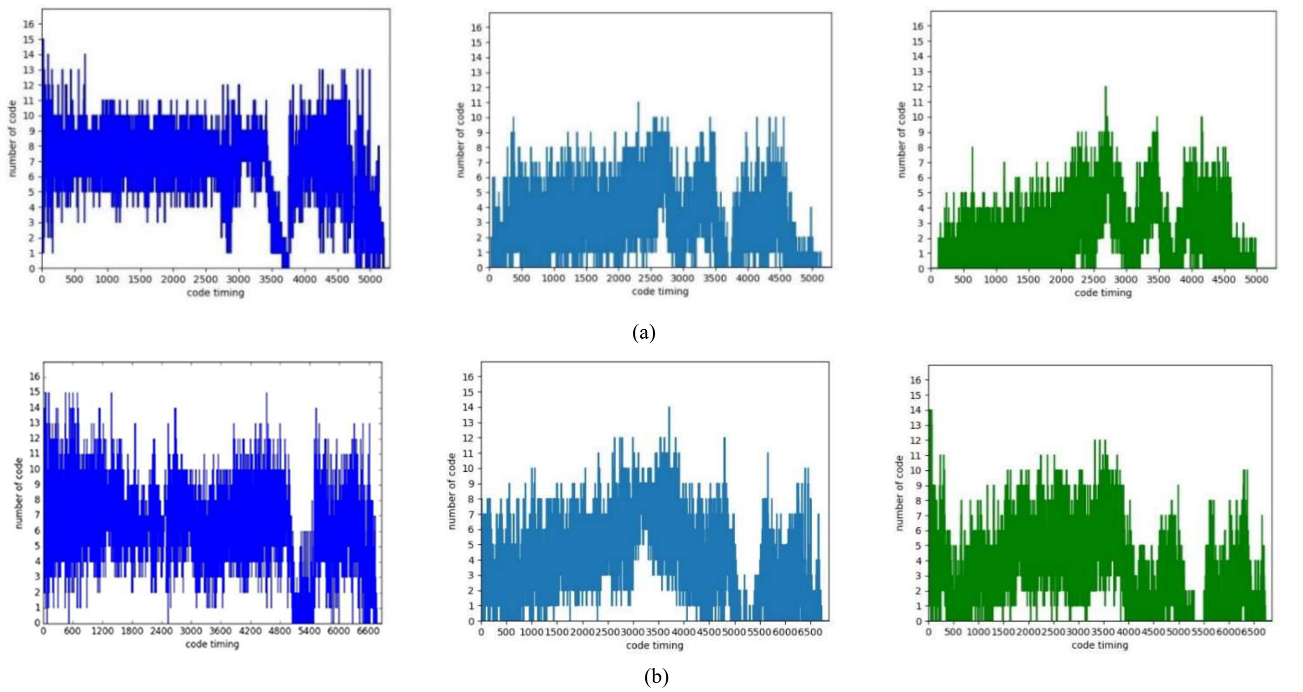


FIGURE 11. **(a) Result of the proposed method with each PE/500kV. (b) Result of [20] with each PE/500kV.**

TABLE 5. **Makespan for Qther simulation instances.**

Simulation	Proposed	[20]
Xi'an[1]	2427	4661
Xi'an[2]	2284	4054
Guangzhou[1]	3990	6347
Guangzhou[2]	3822	6215

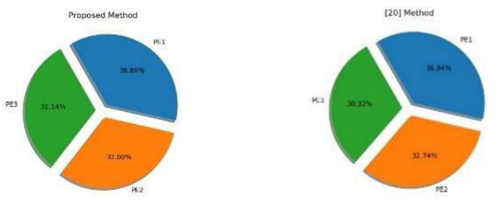


FIGURE 12. **Load balancing of 220kV simulation.**

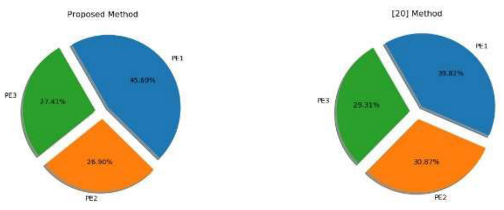


FIGURE 13. **Load balancing of 500kV simulation.**

computing variables, the percentage of communication usage is lower, the amount of reading and writing is less, and the final completion time is significantly reduced. The above makespan statistics confirm our assumption.

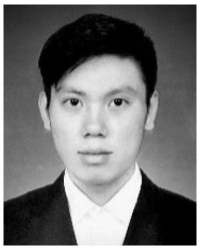
VI. CONCLUSION

This paper proposes a DRL-based task scheduling algorithm to optimize the performance and efficacy of FRTDS with power system simulations. It introduces the block concept and the assumption that unbalanced arrangements cause blocks. A cost model is combined with Q-learning as a Q function to evaluate task choice benefits under certain states with limited tasks to choose. It addresses the importance of resources in modeling and makes resources variables in the model, but not constraints. With the FRTDS our laboratory built, the proposed algorithm is implemented and its assumptions are verified by comparative analysis. With a more balanced strategy, the proposed strategy gains more long-term rewards, which performs as short end-simulation time in one step. We provide resource usage records to support our assumption. We could refer the communication influence to the unbalanced storage and usage of resource in performance as assumption.

Some refinements can be made in future work. Our Q-learning method relies on stored experience, which requires much memory and affects execution speed. We have found more similarities between FRTDS and the A3C structure in parallel computing. We will try to make our DRL process perform with better accuracy and efficacy. We also find that detailed components models and more specific synthetic cost construction for certain FRTDS design could make our results better and we will have more research on maintaining lower step length with detailed models in real time simulations. Our model quantifies the hardware design features and resource allocation as variables. However, the hardware design may change due to expansion of the platform or a new simulation platform. The hardware does affect the construction of SC and PC. We use common indicators like storage, address, and I/O usage to build a model that guarantees swift migration between platforms. However, to use more specific cost models based on certain FRTDS designs may shorten the makespan.

REFERENCES

[1] E. D. Delgado, J. J. R. Panduro, E. C. B. Álvarez, F. J. E. Jurado, and E. F. G. Frías, "Low cost DSP-based educational embedded platform for real-time simulation and fast implementation of complex systems in Simulink," *Comput. Appl. Eng. Edu.*, vol. 27, no. 4, pp. 955–970, Jul. 2019.
[2] R. AhmadiAhangar, A. Rosin, A. N. Niaki, I. Palu, and T. Korõtko, "A review on real-time simulation and analysis methods of microgrids," *Int. Trans. Electr. Energy Syst.*, vol. 29, no. 11, 2019, Art. no. e12106.
[3] V. H. Nguyen, T. L. Nguyen, Q. T. Tran, and Y. Besanger, "Synchronization conditions and real-time constraints in co-simulation and hardware-in-the-loop techniques for cyber–physical energy system assessment," *Sustain. Energy, Grids Netw.*, vol. 20, Dec. 2019, Art. no. 100252.
[4] M. Milton, A. Benigni, and A. Monti, "Real-time multi-FPGA simulation of energy conversion systems," *IEEE Trans. Energy Convers.*, vol. 34, no. 4, pp. 2198–2208, Dec. 2019.
[5] H. Bai, C. Liu, A. K. Rathore, D. Paire, and F. Gao, "An FPGA-based IGBT behavioral model with high transient resolution for real-time simulation of power electronic circuits," *IEEE Trans. Ind. Electron.*, vol. 66, no. 8, pp. 6581–6591, Aug. 2019.
[6] O. Nzimako, C. Jegues, and Y. Zhang, "Dynamic average modelling of renewable generation sources for real time simulation," *J. Eng.*, vol. 2019, no. 18, pp. 4785–4787, Jul. 2019.
[7] C. Haihua and T. Xinhuai, "A load-balance based resource-scheduling algorithm under cloud computing environment," in *New Horizons in Web-Based Learning—ICWL 2010 Workshops: STEG, CICW, WGLBWS, and IWKDEWL* (Lecture Notes in Computer Science), vol. 6537. 2011, pp. 85–90.
[8] B. Aditi, B. Tarun, and K. Pratyay, "A novel genetic algorithm based scheduling for multi-core systems," in *Proc. 2nd Int. Conf. Smart Innov. Commun. Comput. Sci. (ICSICCS)*, 2018, pp. 45–54.
[9] J. Qian, C. Liu, D. Miao, and X. Yue, "Sequential three-way decisions via multi-granularity," *Inf. Sci.*, vol. 507, pp. 606–629, Jan. 2020.
[10] Y. Xie and J. Wu, "Multi-objective constraint task scheduling algorithm for multi-core processors," *Cluster Comput.*, vol. 22, no. 3, pp. 953–964, Sep. 2019.
[11] H. Sun, R. Elghazi, A. Gainaru, G. Aupy, and P. Raghavan, "Scheduling parallel tasks under multiple resources: List scheduling vs. Pack scheduling," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 194–203.
[12] J. P. B. Mapetu, Z. Chen, and L. Kong, "Low-time complexity and low-cost binary particle swarm optimization algorithm for task scheduling and load balancing in cloud computing," *Appl. Intell.*, vol. 49, no. 9, pp. 3308–3330, 2019.
[13] R. Mirzahosseini, R. Iravani, and Y. Zhang, "An FPGA-based digital real-time simulator for hardware-in-the-loop testing of traveling-wave relays," *IEEE Trans. Power Del.*, early access, Mar. 5, 2020, doi: 10.1109/TPWRD.2020.2972737.
[14] R. Mirzahosseini and R. Iravani, "Small time-step FPGA-based real-time simulation of power systems including multiple converters," *IEEE Trans. Power Del.*, vol. 34, no. 6, pp. 2089–2099, Dec. 2019.
[15] C. Yang, Y. Xue, X.-P. Zhang, Y. Zhang, and Y. Chen, "Real-time FPGA-RTDS co-simulator for power systems," *IEEE Access*, vol. 6, pp. 44917–44926, 2018.
[16] Y. Chen and V. Dinavahi, "Multi-FPGA digital hardware design for detailed large-scale real-time electromagnetic transient simulation of power systems," *IET Gener., Transmiss. Distrib.*, vol. 7, no. 5, pp. 451–463, May 2013.
[17] B. Zhang, D. Zhao, Z. Jin, and Y. Wu, "Multivalued coefficient prestorage and block parallel method for real-time simulation of microgrid on FRTDS," *Energies*, vol. 10, no. 9, p. 1248, Aug. 2017.
[18] B. Zhang, S. Fu, Z. Jin, and R. Hu, "A novel FPGA-based real-time simulator for micro-grids," *Energies*, vol. 10, no. 8, p. 1239, Aug. 2017.
[19] B. Zhang, Y. Wu, Z. Jin, and Y. Wang, "A real-time digital solver for smart substation based on orders," *Energies*, vol. 10, no. 11, p. 1795, Nov. 2017.
[20] B. Zhang, R. Hu, S. Tu, J. Zhang, X. Jin, Y. Guan, and J. Zhu, "Modeling of power system simulation based on FRTDS," *Energies*, vol. 11, no. 10, p. 2749, Oct. 2018.
[21] B. Zhang, X. Jin, S. Tu, Z. Jin, and J. Zhang, "A new FPGA-based real-time digital solver for power system simulation," *Energies*, vol. 12, no. 24, p. 4666, Dec. 2019.



Y. GUAN was born in Dalian, Liaoning, China, in 1995. He received the B.S. degree in electrical engineering from Southwest Jiao Tong University, Chengdu, China, in 2018. He is currently pursuing the M.S. degree in electrical engineering from Tianjin University, Tianjin, China. His research interests include set in power system real-time simulation, software computing as well as neural network decision-making. He has published iSPEC 2019 conference paper in the IEEE Xplore. He receive the first prize of the 2019 HUAWEI National Graduate Students Math Modeling Cup with question F, which he took the coding job and accomplished our team's greedy serial route generating algorithm for UAV. The code can be seen in my GitHub repo: https://github.com/Redbaron0601/2019-F-Huawei-Math-Modeling-Competition.git

BD. ZHANG was born in Changshu, Suzhou, China, in 1959. He received the B.S. degree in electrical engineering from the Huanan University of Technology, Guangdong, in 1982, and the M.S. degree in electrical engineering from Tsinghua University, Beijing, China, in 1988.

From 1982 to 1985, he was a Teacher in mechanical academy with Jiangnan University, Wuxi. Since 1988, he has been a Teacher and then a Professor with the Electrical Engineering Department, Tianjin University, China. He is the Associate Editor of *Journal of Power System and Automation*. He is also the author of more than 20 articles and eight innovations and the book *Foundation of Intelligent Information Processing Technology*. His research interests include real-time power system simulation, optimal operation and control of power systems, power quality monitoring and control, monitoring and control of high-voltage power equipment, and hardware-in-the-loop test implementation.

Z. JIN was born in Dezhou, Shandong, China, in 1992. He received the bachelor's degrees majoring in automation and electrical engineering from the Tianjin University, in 2015, where he is currently pursuing the Ph.D. degree in electrical engineering.

He is the author of six articles, and three articles are under review. His research interests include parallel computing, simulation study as well as FPGA computing and power control of HVDC flexible systems, and real time simulation of power systems.