Computer Science and Information Systems 21(4):1335–1357 https://doi.org/10.2298/CSIS231129035C

A Method for Solving Reconfiguration Blueprints Based on Multi-Agent Reinforcement Learning

Jing Cheng[1,⋆], Wen Tan[1], Guangzhe Lv[2], Guodong Li[3], Wentao Zhang[4], and Zihao Liu[5]

[1] School of Computer Science and Engineering, Xi'an Technological University, Xi'an 710016, China
chengjing@xatu.edu.cn, tanwen@st.xatu.edu.cn

[2] Xi'an Institute of Aeronautical Computing Technology (XICT), Xi'an 710065, China
guangzhe_lv@163.com

[3] Aviation Industry First Aircraft Design and Research Institute, Xi'an 710089, China
guodonglee@163.com

[4] School of Software, Northwestern Polytechnical University, Xi'an 710129, China
wentaoz1223@gmail.com

[5] Leihua Electronic Technology Research Institute of Aviation Industry Corporation of China (AVIC), Wuxi, China
2020264414@mail.nwpu.edu.cn

Abstract. Integrated modular avionics systems primarily achieve system fault tolerance by reconfiguring the system configuration blueprints. In the design of manual reconfiguration, the quality of reconfiguration blueprints is influenced by various unstable factors, leading to a certain degree of uncertainty. The effectiveness of reconfiguration blueprints depends on various factors, including load balancing, the impact of reconfiguration, and the time required for the process. Solving high-quality reconfiguration configuration blueprints can be regarded as a type of multi-objective optimization problem. Traditional algorithms have limitations in solving multi-objective optimization problems. Multi-Agent Reinforcement Learning (MARL) is an important branch in the field of machine learning. It enables the accomplishment of more complex tasks in dynamic real-world scenarios through interaction and decision-making. Combining Multi-Agent Reinforcement Learning algorithms with reconfiguration techniques and utilizing MARL methods to generate blueprints can optimize the quality of blueprints in multiple ways. In this paper, an Improved Value-Decomposition Networks (VDN) based on the average sequential cumulative reward is proposed. By refining the characteristics of the integrated modular avionics system, mathematical models are developed for both the system and the reconfiguration blueprint. The Improved VDN algorithm demonstrates superior convergence characteristics and optimization effects compared with traditional reinforcement learning algorithms such as Q-learning, Deep Q-learning Network (DQN), and VDN. This superiority has been confirmed through experiments involving single and continuous faults.

Keywords: Integrated modular avionics system, Multi-Agent Reinforcement Learning, reconfiguration blueprint, multi-objective optimization problem.

⋆ Corresponding author

1. Introduction

The modular avionics system offers several advantages, including system integration, a layered structure, network unification, scheduling flexibility, and centralized maintenance. It finds widespread applications in aerospace fields, including fighter aircraft, civil aircraft, and satellites [1,2,3]. With the continuous expansion of the functionality of integrated electronic systems, the probability of system failures has also increased dramatically, posing significant challenges to the safety and reliability of the systems. Consequently, fault-tolerant mechanisms for integrated electronic systems [4,5,6] are critical tasks.

Reconfiguration is a crucial mechanism for fault tolerance in integrated electronic systems. Reconfiguration in integrated electronic systems refers to the process of reallocating system resources and loading system software to ensure complete system functionality when faults occur in system partitions or software resources [7]. The reconfiguration of integrated electronic systems is primarily achieved through the generation of reconfiguration blueprints [8,9]. There are two main approaches to generating reconfiguration blueprints: manual design and traditional algorithms [10,11]. Manual design is influenced by practical experience and individual differences, while traditional algorithms emphasize feasibility but have limitations in solving multi-objective optimization problems [12,13,14]. Multi-agent reinforcement learning is a crucial branch of machine learning that can handle more complex tasks in dynamic real-world scenarios [15,16,17]. Generating reconfiguration blueprints using a multi-agent reinforcement learning approach allows for optimizing blueprints from various perspectives. Therefore, this paper proposes an intelligent method for generating reconfiguration blueprints for integrated electronic systems based on multi-agent reinforcement learning.

The main contributions of this paper include:

1) The establishment of mathematical models for integrated electronic systems and reconfiguration blueprints. This involves creating models for system resources, system partition, and system software based on the characteristics of integrated electronic system reconfiguration. Additionally, models for system faults, reconfiguration configuration blueprints, and system migration are developed considering the characteristics of reconfiguration blueprints.

2) The proposal of an Improved VDN algorithm based on average sequential cumulative rewards. Grounded in the VDN algorithm in multi-agent reinforcement learning, this algorithm is designed in conjunction with a comprehensive electronic system. It includes reconfiguration states, actions of agent, overall and local reward functions, and algorithmic strategies. The introduction of an adaptive exploration strategy based on the average sequence cumulative reward enhances the efficiency and accuracy of algorithmic reconfiguration. This builds upon the original exploration strategy of the VDN algorithm. Finally, an analysis and verification of the results of the Improved VDN algorithm are conducted, comparing it with reconfiguration algorithms implemented based on Q-learning [18], DQN [19], and VDN [20], confirming the superiority of the proposed Improved VDN algorithm.

2. Related Works

Reconfiguration techniques play a crucial role in maintaining the security, reliability, survivability, maintainability, and scalability of integrated modular avionics systems, holding substantial practical significance and economic value. Extensive research has been conducted on reconfiguration techniques by professionals both domestically and internationally. Housseyni et al. [15] proposed a distributed reconfiguration method based on a multi-agent model. The method considers the functional module as the smallest unit of reconfiguration, representing each device as an agent. Each agent undergoes reconfiguration at three levels, with the first level responsible for adding and removing functional modules, the second level for integrating functional modules, and the third level for updating functional modules with data. The method ensures the security of distributed reconfiguration through agent communication in eXtensible Markup Language (XML) format. Zhou et al. [21] introduced a functional architecture framework for the Distributed Integrated Modular Avionics (DIMA) platform, utilizing the Architecture Analysis and Design Language (AADL) action model to construct the functional action characteristics of DIMA's dynamic reconfiguration. This framework provides a design basis for implementing DIMA's dynamic refactoring. Saadi A et al. [22] proposed a methodology for validating software refactoring using the Communicating Sequential Processes (CSP) language, refactoring techniques, and the FDR model, aiming to guide the process of software refactoring. Ensuring the consistency of dynamic software refactoring involves conducting a comprehensive analysis before making any modifications to the software. The ability to safely execute software refactoring at runtime is crucial for the feasibility of integrated modular avionics system refactoring. Chen et al. [23] proposed an AADL model based on the reconfiguration process of an integrated modular avionics system. Using the accessibility analysis method of Petri nets, the danger of IMA reconfiguration function is analyzed. This method overcomes the weaknesses of the traditional static analysis method, making it more capable of analyzing hazardous behavior during runtime. It provides a solution to the state explosion problem that arises during reachability analysis.

Literature [24,25] has confirmed the effectiveness of reinforcement learning in optimizing planning. Zhang et al. [26] applied sequential game multi-agent reinforcement learning to the field of integrated modular electronic system reconfiguration. They utilized a policy gradient Monte Carlo search tree algorithm incorporating bias estimation to expedite the convergence of the conventional algorithm. However, the method does not address the adaptation of the exploration factor during the competition and cooperation phases of the agent. Liu et al. [27] proposed a reliability analysis method based on the AADL model, using Petri nets to analyze the reliability of the reconfiguration method. This method assists in designing reconfiguration models at the early stage of integrated modular avionics system development, ultimately improving the reliability of the entire integrated modular avionics system. Hollow et al. [28] proposed a reconfiguration scheme for integrated modular avionics systems based on a simulated annealing algorithm. The scheme finds an equivalent reconfiguration scheme with a search function for the integrated modular avionics system by calculating the fitness. While it has theoretical significance, the method exhibits certain defects, such as the inability to meet timeliness. Cui et al. [10] proposed a reconfiguration technique for integrated modular avionics systems based on distributed techniques. This technique enables the system to adapt to the reconfiguration task, quickly detect and locate faults, and establish a blueprint for reconfiguration. Suo et al. [12] introduced an integrated modular avionics system reconfiguration safety detection technique based on STPA. The method focuses on coordinating human-computer interactions and detecting system failures based on the severity of the failure, human factors, and time constraints. The analysis results can be utilized for system development, system operation, and project revision, ensuring the safety of reconfigurable integrated modular avionics systems.

3. Research Methodology

3.1. Integrated Modular Avionics System Modeling

The integrated modular avionics system consists of hardware resource modules, operating system modules, and software modules. A single module satisfies functional independence, while multiple modules provide isolation. Additionally, the overall system ensures security and reliability [29,30,31]. The simplified schematic of the integrated modular avionics system is shown in Figure 1.

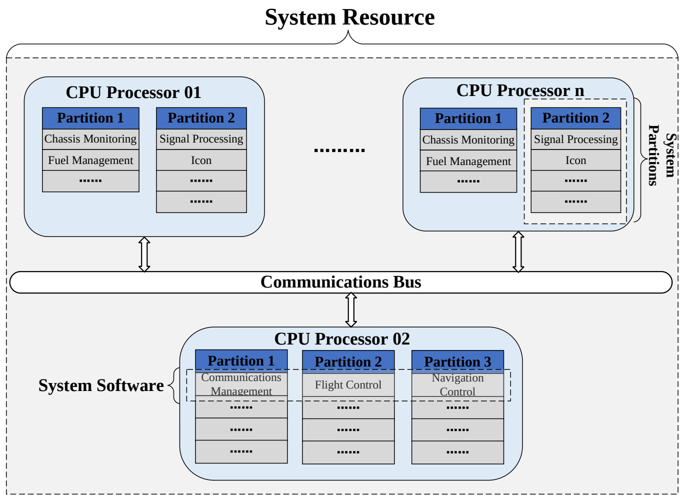

Fig. 1. Schematic diagram of the simplified model of the integrated modular avionics system

System resource model. In this paper, the system hardware resources are abstracted into a system hardware module. This module should include attributes such as CPU resource list, partition resource list, software resource list, and operation status to realize the function of easy maintenance and management of partition collection. The hardware resources are represented using a quaternion model as shown in Equation (1).

$$\begin{gathered} Resource = \langle CPU_{list}, Patition_{list}, Application_{list}, state \rangle \\ CPU_{list} = \{CPU_1, CPU_2, ..., CPU_S\} \\ Partition_{list} = \{Partition, Partition_2, ..., Partition_M\} \\ Application_{list} = \{App_1, App_2, ..., App_N\} \\ state = \{0, 1\} \end{gathered} \tag{1}$$

Where $Resource$ denotes the system resources. CPU_{list} denotes the set of central processor resources. $Partition_{list}$ denotes the set of partitions configured on the processor. $Application_{list}$ denotes the set of software resources configured on partitions. $state$ denotes the health state of the system resources. 0 is the system resource policy. 1 is the system resource failure. S denotes the total number of system processors. M denotes the total number of system partitions. N denotes the total number of system software applications.

System partition model. The modeling of system partitions is carried out in terms of both temporal and spatial isolation. Combining the principles of temporal scheduling and spatial mapping of partitions, the model for constructing partitions for temporal and spatial attributes is shown in Equation (2) below.

$$\begin{gathered} Partition_i = \langle time_{start}, time_{duration}, \\ mem_{offset}, memsize_{Partition_i}, Application_{list}, state \rangle \end{gathered} \tag{2}$$

Where $time_{start}$ denotes the start execution time of the partition. $time_{duration}$ denotes the duration of execution of the partition. mem_{offset} denotes the offset address of the partition's virtual memory. $memsize_{Partition_i}$ denotes the size of the virtual memory of partition i. $Application_{list}$ denotes the set of applications deployed in the partition. $state$ indicates the health status of the system partition, with 0 being a healthy partition and 1 being a faulty partition.

System software model. The system software model is constructed in terms of both software spatial resources and software temporal resources. The hexadecimal model of software resource modeling is depicted in Equation (3).

$$App_{ij} = \langle pid, rank, run_{cycle}, WCET, time_{deadline}, mem, state \rangle \tag{3}$$

Where App_{ij} denotes the application software model, where the partition number is i and the software number is j. pid denotes the process ID of the application software. $rank$ denotes the task priority of the application software. run_cycle denotes the execution cycle of the application software. $WCET$ denotes the worst-case execution time of the process. $time\_deadline$ denotes the application software deadline. mem indicates the space resources required by the application software. $state$ indicates the health status of the system software, with 0 being software health and 1 being software failure.

3.2. Reconfiguration Blueprint Model

System Failure Model. There are two main types of faults in the integrated modular avionics system: processor faults and processor partition faults [32,33]. When any one of these faults occurs, or when multiple faults occur in succession, the integrated modular avionics system enters a fault state [34]. This paper presents a model called the System Fault Abstraction Model (SFAM) for abstracting system faults. To uniquely identify faults and their occurrence locations, two attributes, namely system fault type and system fault location, are required to create a two-tuple model for the system fault model (4).

$$Fault = \langle type, location \rangle \tag{4}$$

Where $Fault$ denotes the system failure model, $type$ denotes the system failure type, and $location$ denotes the system fault location.

It is necessary to establish the following constraints to distinguish between the two types of faults and their respective locations in the integrated modular avionics system.

$$Fault_{ij} = \begin{cases} type =\subseteq \{F_i, F_{ij}\}, \, (1 \le i \le S, 1 \le j \le M) \\ location_i = C_i, \quad\;\; type = F_i \\ location_{ij} = C_i P_j, \, type = F_{ij} \end{cases} \tag{5}$$

F_i indicates that a processor failure has occurred in the system, and the faulty processor is numbered i. F_{ij} indicates that a processor partition failure has occurred in the system. The faulty partition is located in the jth partition of the CPU number i. C_i indicates the location of a system processor failure with faulty processor number i. C_iP_j indicates the location of a system processor partition failure.

Reconfiguring the configuration blueprint model. The reconfig blue print (RBP) model consists of the following set of five Equations (6) and (7).

$$\begin{gathered} RBP = \langle CPU_{list}, Partition_{list}, Application_{list}, Fault_{list}, Map \rangle \\ Fault_{list} = Fault_1, Fault_2, \ldots, Fault_K \end{gathered} \tag{6}$$

$$\begin{gathered} Map = \left\{ application_i \xrightarrow{CPU_s} Partition_j \right\} \\ (i \in [1, |Application_{list}|] \, , j \in [1, |Partition_{lsit}|]) \end{gathered} \tag{7}$$

The reconfiguration blueprint model above illustrates the mapping relationship between processors, partitions, and software in the reconfiguration blueprint. For example: $Map = \left\{ App_1 \xrightarrow{C_1} Partition_1 \right\}$ denotes that the application software numbered 1 is deployed on the partition numbered 1. The partition numbered 1 is deployed on the processor numbered 1. $Fault_{list}$ is a list of faults, and the number of faults is K.

Refactoring Migration Blueprint Model. Reconfigure Migrate Blueprint (RMB) consists of a reconfigured initial state, a set of reconfigured intermediate states, a reconfigured end state, and their corresponding transfer relationships. The Reconfigure Migration Blueprint model definition consists of five tuples, as shown in Equation (8) below.

$$\begin{gathered} RMB = \langle S_{init}, S_{end}, Fault_{list}, S_{list}, Move \rangle \\ S_{list} = \{S_1, S_2, ..., S_n\} \\ Move = \left\{ S_{src} \xrightarrow{Fault} S_{des} \right\} \end{gathered} \tag{8}$$

Where RMB denotes the refactoring migration blueprint model. S_{init} denotes the refactoring initial state. S_{end} denotes the refactoring end state. S_{list} denotes the set of refactoring intermediate states. $Move$ denotes the refactoring transfer relation. S_{src} denotes the source state in the refactoring transfer relation. S_{des} denotes the destination state of the refactoring transfer relation.

Blueprint Quality. To improve the quality of the filtered feasible reconfiguration blueprints, this paper introduces optimization objectives to assess their quality.
A.Load balancing

Rationalizing the allocation of resources across multiple partitions in an integrated modular avionics system will improve the quality of reconfiguration blueprints. In this paper, we use the term ”load” to refer to the evenly distributed allocation of partition resources. Load is calculated as the weighted sum of CPU utilization and memory utilization for each partition. The calculation formula for establishing load balancing is provided in Equation (9) below.

$$\begin{cases} LB = 1 - 2 * \sqrt{\frac{1}{M} \sum_{i=1}^{M} \left(Load_{Pi} - \overline{Load} \right)^2} \\ Load_{pj} = \mu_1 * C_{use}^{P_j} + \mu_2 * M_{use}^{P_j} \\ C_{use}^{P_j} = \frac{\sum_{j=1}^{|Application_{list}|} WCET_{app_{ij}}}{Partition_{durtime}} \\ M_{use}^{P_j} = \frac{\sum_{j=1}^{|Application_{list}|} mem_{app_{ij}}^{Partition_i}}{memsize_{Partition_i}} \\ \mu_1 + \mu_2 = 1 \end{cases} \tag{9}$$

where LB denotes load balancing and $Load_{P_j}$ denotes denotes the load on the partition numbered i. $\overline{Load}$ denotes the average resource load on all partitions. μ_1 denotes the weight of CPU utilization, and denotes the weight of memory utilization. $C_{use}^{P_j}$ denotes the partition p_j of CPU utilization, the $M_{use}^{P_j}$ denotes the weight of the partition's p_j of the partition. The memory utilization of the partition.
B.Reconfiguration fault tolerance

Refactoring Fault Tolerance (RFT) is used to measure the impact of refactoring blueprints on the system. The success rate of migrating the software set to the remaining available partitions is determined by the weighted proportion of the software that is successfully migrated. This paper emphasizes the importance and criticality of integrated modular avionics system software. The integrated modular avionics system software is categorized into five levels. Level 5 is the highest level of importance, while Level 1 is the lowest level of importance. According to the definition of migration success rate, the indicator for migration success rate is established as shown in Equation (10) below:

$$RFT = \frac{\sum_{i=1}^{n_M} G_{M_i}}{\sum_{i=1}^{N_M} G_{M_i}} \tag{10}$$

C.Reconfiguration time rate

During the reconfiguration migration process, each software that requires reconfiguration migrates in parallel, following the sequence outlined in the reconfiguration blueprint. The individual processor refactoring time is defined as the total time required to reload all the software in the respective partition of the processor. The lower the percentage of time spent on refactoring, the higher the fault tolerance of the refactoring blueprint, and the lesser impact it has on system functionality. The reconfiguration time rate metric is established as shown in Equation (11) below.

$$\begin{cases} RTR = 1 - \frac{Max(T_{CPU_s})}{T_{max}} \\ T_{CPU_s} = \sum_{i=1}^{n} T_{App_i} \end{cases} \tag{11}$$

RTR denotes the reconfiguration time. T_{CPU_s} denotes the reconfiguration recovery time for processor numbered s. T_{max} denotes the maximum reconfiguration time. N_{re} denotes the processor c_k number of software in the processor that needs to be reconfigured. The T_{M_i} denotes the application software M_i reconfiguration migration time of the application software.

D.Load optimization function

To allocate more resources for future reconfiguration actions and enhance the success rate and efficiency of reconfiguration, a load optimization function is proposed. It is shown in Equation (12) below.

$$Minf_1 = \sum_{i=1}^{M} \sum_{j=1}^{N} \mu_1 C_{use}(P_i, App_{ij}) + \mu_2 M_{use}(P_i, App_{ij})$$
$$\begin{cases} \mu_1 + \mu_2 = 1 \\ \forall Mem_{use}(P_i, App_{ij}) \leq Mem_{max} \\ \forall C_{use}(P_i, App_{ij}) \leq CPU_{max} \end{cases} \tag{12}$$

f_1 denotes the load optimization function. $C_{use}(P_i, App_{ij})$ denotes the software to be migrated App_{ij} to be migrated to the partition P_i after being migrated to a partition. $\mu_2 M_{use}(P_i, App_{ij})$ denotes the CPU utilization of the software to be migrated App_{ij} to be migrated to the partition P_i memory utilization after being migrated to a partition. $Mem_{use}(P_i, App_{ij})$ represents the memory utilization of the software to be migrated, App_{ij}, after it has been migrated to partition P_i. Mem_{max} indicates the maximum memory available to the processor. And $C_{use}(P_j, D_i)$ Indicates the maximum memory available on the processor for the software to be migrated App_{ij} to be migrated to the partition P_i CPU resource consumption after being migrated to the partition.CPU_{max} indicates the maximum CPU resources available to the processor.

E.Multi-objective optimization function

The multi-objective optimization function is established as shown in Equation (13) below.

$$Maxf_2 = \lambda_1 LB + \lambda_2 RFT + \lambda_3 RTR$$
$$\begin{cases} 0 < \lambda_i < 1, i \in [1, 3] \\ \lambda_1 + \lambda_2 + \lambda_3 = 1 \end{cases} \tag{13}$$

Where f_2 denotes the multi-objective optimization function, λ_1 denotes the load balancing weights, and λ_2 denotes the reconfiguration fault tolerance weights, and λ_3 denotes the reconfiguration time share weights.

3.3. Reconfiguration method based on Improved VDN algorithm

Overview. In this paper, the algorithm for generating intelligent reconfiguration blueprints incorporates the design concepts of the VDN multi-agent reinforcement learning algorithm. It refines the characteristics of the reconfiguration task of a comprehensive electronic system and incorporates them into the design of agent, state design, action space design, and payoff function design. Specifically, the software that requires reconfiguration is referred to as an agent. The decision sequences are organized based on the software's level of importance, which ranges from high to low. The partition state is represented by the weighted sum of time and space resources consumed by all partitions. The partitioned states, along with the observations made by the agent about their states, are utilized as the states in the reinforcement learning process. The assignment of each agent to an available partition or scheduling failure is abstracted as the action space. For the exploration and exploitation strategy, this paper adopts a strategy based on the cumulative reward of the average sequence. The cumulative rewards in the environment information are utilized to allow the algorithm to adaptively adjust the exploration factor during the training phase. For the reward function, two separate functions are designed for the agent competition phase and the agent cooperation phase. It is used to achieve the goal of balancing the optimization of agent systems and overall efficiency. When the algorithm reaches the termination round, the reconfiguration configuration blueprint is obtained.

Environmental design. Each software that requires reconfiguration is defined as a standalone agent. Each agent can independently sense the state of the environment, take actions, and receive feedback from the environment. All the agents compete and cooperate within the integrated modular avionics system environment. In this paper, a state matrix is used to describe the reconfiguration state of the integrated modular avionics system.

In conjunction with the integrated modular avionics system, the CPU information, partition information, and application software information are enumerated first. Let's consider an integrated modular avionics system that consists of three CPUs and six partitions. Each partition has 50 ms of available time slice resources and 50 KB of available memory resources. During the process of refactoring, the sequence of software refactoring significantly impacts the quality of the refactoring and the migration plan. This paper decomposes the fault granularity at the software level when defining the refactoring state. In other words, CPU2 faults are divided into software M3 faults and software M4 faults. At this point, two agents, M3 and M4, are generated according to the algorithm's design. As shown in Figure 2, the state matrix of agent M3 has been established.

The specific meaning of the state matrix is illustrated in the Equations. (14) and (15) below. The state matrix is divided into two rows. The first row represents the states of all partitions of the integrated modular avionics system, which correspond to the current observations of the agent in its environment. The weighting coefficients for the normal partitions are 0.5 for both the operation cycle and the operation memory, and -0.5 for the faulty partitions. The second row represents the state of the software that needs to be reconfigured. This pertains to the observation of agent within the framework of multi-agent reinforcement learning [35,36,37]. The observations of the agent about the environment

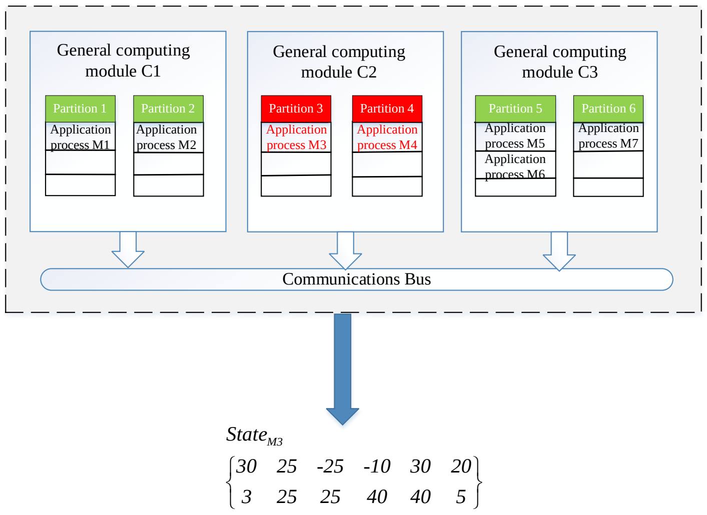

Fig. 2. M3 Reconfiguration State Chart

are combined with their observations of themselves to form comprehensive insights.

$$State_{(i,j)} = \begin{cases} ParInfo_{p_j} & (i = 0, 0 < j < m) \\ App_{index} & (i = 1, j = 0) \\ App_{dur} & (i = 1, j = 1) \\ App_{mem} & (i = 1, j = 2) \\ App_{WD} & (i = 1, j = 3) \\ App_{deadline} & (i = 1, j = 4) \\ App_{rank} & (i = 1, j = 5) \\ 0 & else \end{cases} \tag{14}$$

$$ParInfo_{p_j} = \begin{cases} 0.5 * Duruse_{_{p_j}} + 0.5Memuse_{_{p_j}} & (0 < j < m, P_j trouble - free) \\ -0.5 * Duruse_{_{p_j}} - 0.5Memuse_{_{p_j}} & (0 < j < m, P_j trouble) \end{cases} \tag{15}$$

Where $State_{(i,j)}$ denotes the state matrix. m denotes the number of partitions. $ParInfo_{Pj}$ denotes the information of partition j. App_{index} denotes the software number. App_{dur} denotes the software runtime period. App_{mem} denotes the software runtime memory. App_{WD} denotes the worst runtime of the software. $App_{deadline}$ denotes the software cutoff time. App_{rank} denotes the software importance level. $Duruse_{Pj}$ denotes the remaining time slice resources of partition j. and $Memuse_{Pj}$ denotes the remaining memory resources of partition j.

In this paper, the reconfiguration of agent follows the software priority order, indicating that software with high priority is reconfigured before software with low priority. When the agent with high priority executes an action, it affects the environment and other prioritized observations. In this example, the agent M3 has a priority of 5, while the agent M4 has a priority of 4. M3 executes the refactoring before M4. After executing the reconfiguration action, the agent M3 migrates the application to partition 2. But partition 2 does not have any available resources at the moment. When the agent M4 executes the action, its state matrix changes, as depicted in Figure 3.

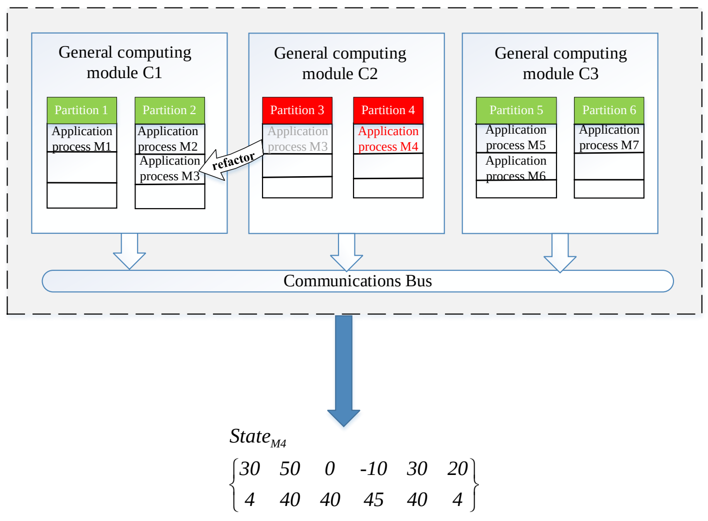

Fig. 3. M4 reconfiguration state diagram

Agent Action Design. In this paper, an individual action of an agent is defined as the process of scheduling the current software to be reconfigured for a specific available partition. The set of all executable discrete scheduling actions constitutes the action space. The specific definition is shown in Equation (16). When there are m available partitions, the size of the current action space is m. It contains m successful scheduling actions. That is, the software that needs to be reconfigured can be scheduled for any of the available partitions.

$$
\begin{gathered}
Action_i = (Act_0, Act_1 ... Act_{m-2}, Act_{m-1}) \\
Act_j = \begin{cases} 1 \; j = k, k \in [0, m-1] \\ 0 \qquad\qquad else \end{cases} \\
ActionSpace \subseteq \{Action_i\} \qquad (i \in [0, m-1])
\end{gathered} \tag{16}
$$

m denotes the number of partitions. $Action_i$ denotes the i-th action. Act_j denotes the j-th action component, and $ActionSpace$ denotes the set of action spaces.

Exploratory strategies based on average sequential cumulative rewards. During the process of algorithm training, the agent selects actions based on a strategy of exploration and exploitation. In the traditional VDN algorithm, the strategy is defined as shown in Equation (17) below.

$$\pi\left(a|s\right)=\begin{cases}1-\varepsilon & a=argmaxQ\left(s,a\right)\\ \varepsilon & a=random\left(a\right)\end{cases} \tag{17}$$

The algorithm generates a random number specifically when it is executed. When the random number is less than the exploration coefficient, the algorithm executes the exploration policy. This policy randomly selects an action from the action space for execution. When the random number is greater than 0, the algorithm executes the exploitation strategy, which selects the optimal action based on the existing experience. The traditional strategy is to gradually decrease the value over time. It does not consider the feedback information from the agent during the training process, and it lacks informed action selection.

The algorithm in this paper explores a utilization strategy using an adaptive approach based on the cumulative reward of the average sequence [38]. The exploration factor is dynamically adjusted based on the cumulative reward value of the agent during the training process. In reinforcement learning, the sequence of actions taken by an agent from the initial state to the goal state is crucial. The cumulative reward of the sequence is defined by Equation (18).

$$G_t=\sum_{k=0}^{T}\gamma^k R_{t+k+1} \tag{18}$$

G_t denotes the cumulative serial reward at moment t,γdenotes the reward decay coefficient, and R_{t+k+1} denotes the reward value of the k+1st action at moment t. Therefore, the exploration factor$\varepsilon$and the average sequence cumulative reward are defined as shown in Equation (19) below.

$$\begin{gathered}\varepsilon=\frac{1}{1+\log_2(\bar{G}+1)}\\ \bar{G}=\frac{1}{L}\sum_{i=e-L}^{e-1}G_i\end{gathered} \tag{19}$$

L denotes the number of sequence entries, $\bar{G}$ denotes the average reward of the first L sequences. With the exploration utilization strategy described above, the exploration factor is dynamically adjusted based on the average cumulative reward obtained sequentially. Aggregated environmental feedback information guides the agent in choosing their actions.

Reward function design. During the reconfiguration of an integrated modular avionics system, a new blueprint is generated after performing an application migration operation. The reward function is used to evaluate the impact of the action on the refactoring process. For a positive impact, the reward function provides positive feedback. Increasing the reward value tends to motivate the agent to perform better actions in the next round of training. The reward function provides negative feedback for unfavorable outcomes. Decrease the reward value to discourage the agent from engaging in negative actions. The algorithm in this paper is designed with two reward functions. These functions are used to calculate the cost of actions and the overall reward of the agent.

The cost-reward function of agent actions is primarily applied during the competitive phase of multi-agent reinforcement learning. During this phase, the agents compete with each other for time slices and memory resources in order to achieve higher local payoffs. For an agent system, successful migration will result in the efficient utilization of the allocated time slice and memory resources for each partition. And the consumed resources are inversely proportional to the cost of that action. That is, the smaller the ratio of consumed partition resources to available partition resources, the more favorable the migration of other agents. Once the migration fails, it indicates that the action is unfavorable and penalizes the agent. The final reward function of a single agent is shown in Equation (20).

$$r(s,a)_t^j = \begin{cases} 1-\mu_1 * Cpuuse_{Patition_i}^j - \mu_2 * Memuse_{Patition_i}^j & migration\ successful \\ -1 & migration\ failed \\ \mu_1 + \mu_2 = 1 & \end{cases} \tag{20}$$

$r(s,a)_t^j$ denotes the local reward return value for agent j at time t. $Cpuuse_{Patition_i}^j$ denotes the time slice occupancy of the partition after agent j migrates to partition i. $Memuse_{Patition_j}^i$ denotes the memory occupancy of the partition after agent j migrates to partition i. μ_1 denotes the time slice weighting coefficients, and μ_2 denotes the memory weighting coefficients.

The overall reward function is primarily used in the cooperative phase of multi-agent reinforcement learning. In this phase, the goal of cooperation among the agents is to achieve a higher overall reward. At this time, some individuals may receive a reduced reward in exchange for an overall increase in rewards. This prevents all agents from falling into local optimal solutions by avoiding the use of greedy strategies. The algorithm's explorability and stability have been enhanced. According to the multi-objective optimization constraint metrics in the resource constraints of the reconfiguration blueprint, this paper evaluates the current agent actions based on load balancing, reconfiguration fault tolerance, and reconfiguration time ratio. The overall reward function is shown in Equation (21).

$$R(s,a)_t^j = \begin{cases} \lambda_1 * LB + \lambda_2 * RTR + \lambda_3 * RID & migration\ successful \\ -1 & migration\ failed \end{cases} \tag{21}$$

$R(s,a)_t^j$ denotes the overall reward return value for agent j at time t.

4. Results and Discussion

4.1. Experimental design

In this paper, we have reconfigured the blueprint for the algorithm that generates intelligence, written in the Python language. The parameters of the agent need to be set when training the agent. The parameter settings are shown in Table 1 below.

The network topology structure employs a traditional DQN MLP network, as illustrated in Figure 4 below.

Based on the system migration model of the integrated modular avionics system and the experimental data, the reconfiguration configuration state migration graph shown in

Table 1. Parameter Settings

parameters	parameter value
Training round	20000
Competitive learning rate	0.1
Cooperative learning rate	0.9
Sample size	16
Empirical sample size	2000
Target network update rate	0.01
Memory utilization weights μ_1	0.5
Time-slice utilization weights μ_2	0.5
Starting rounds of cooperation	16000
Average accumulated experience bars L	10

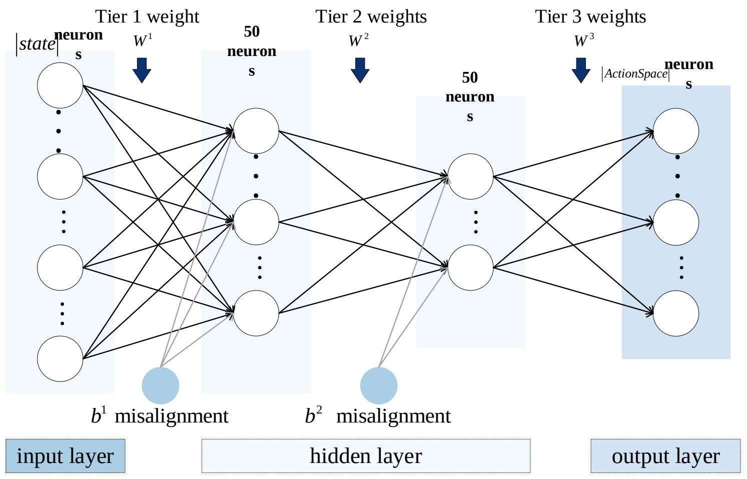

Fig. 4. Schematic of MLP network topology

Figure 5 below has been constructed. Among them, S0 is a blue node representing the initial configuration blueprint. There are eight faults, each of which generates a reconfiguration state. There are eight green nodes corresponding to S1 to S8. S_{end} is a red node representing the final state of the reconfiguration. Two scenarios cause a reconfiguration state to enter the end state. The first reason is that there are no faults that require reconfiguration. The second reason is that the available system resources are insufficient to meet the reconfiguration requirements.

Specifically, the initial state is S0. When the system injects a single fault with fault number ”Fault1,” System Partition 1 fails. The system invokes the intelligent reconfiguration algorithm to generate a new reconfiguration blueprint. The system restores the normal state of S1 by migrating the application and utilizing the reconfiguration blueprint. When the system injects a continuous fault with a fault number of Fault5, a continuous fault occurs in system partitions 1 and 3. At this point, the system proceeds to invoke the intelligent reconfiguration algorithm to create a new reconfiguration blueprint based on S1.

Subsequently, the system executes the reconfiguration process to restore the system to its normal state, S5. Since the consecutive faults for Fault5 have been reconfigured, there are no longer any faults in the system. Therefore, the state is transferred to the reconfiguration end state S_{end}. The remaining steps of the state transfer process are the same as described above.

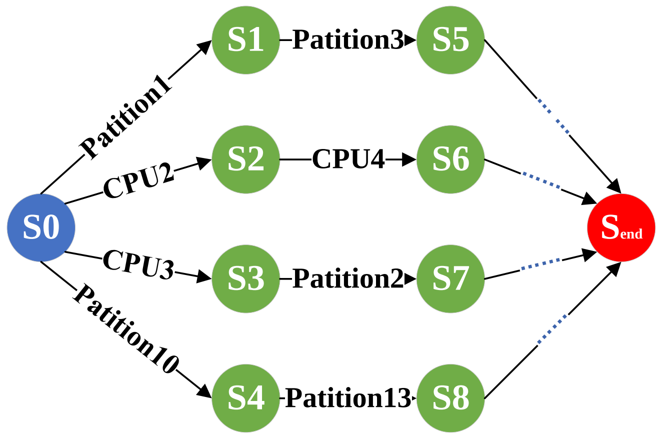

Fig. 5. System Reconfiguration Configuration State Migration Diagram

4.2. Experimental results and analysis

The configuration table is generated for the eight faults mentioned above in this experiment. After the experiment, all the faulty applications were moved to the available partitions. The reconfiguration blueprint meets the requirements for time constraints, space constraints, and uniqueness constraints. The validity of the experimental results is verified.

Parameters affect experimental results. For the parameter impacts of the algorithm, this paper takes the reconfiguration impacts arising from an environment in which fault 1 occurs in an integrated modular avionics system as an example. The impact of the single fault parameter experiment is analyzed. Fault 1 is a typical single fault. That is, 1 partition fails and 1 software needs to be reconfigured. For the aforementioned fault, the experimental results are presented in Table 2 of the Single Fault Parameter Experiment data.

This paper presents an example of the reconfiguration impact in an environment where a Fault6 fault occurs in an integrated modular avionics system. It analyzes the impact of experiments on continuous fault parameters. Fault6 is a typical continuous fault. Three partitions have failed and six software need to be reconfigured. For the above fault, the results of the experiment are shown in Table 3 of the Continuous Fault Parameter Table experiment data below.

Table 2. Single Fault Parameter Experiment Data Sheet

λ_1	λ_2	λ_3	maximum values	convergence value	LB	$RFFT$	MSR
0.45	0.1	0.45	0.941853	0.931086	0.851807	0.940000	1.000000
0.4	0.2	0.4	0.939961	0.931882	0.859705	0.940000	1.000000
0.35	0.35	0.3	0.937844	0.929896	0.859705	0.940000	1.000000
0.25	0.45	0.3	0.942316	0.939148	0.858650	0.940000	1.000000
0.2	0.4	0.4	0.952482	0.946362	0.862412	0.940000	1.000000
0.1	0.45	0.45	0.961241	0.958732	0.887322	0.940000	1.000000

Table 3. Experimental Data Sheet for Continuous Fault Parameters

λ_1	λ_2	λ_3	maximum values	convergence value (math.)	LB	$RFFT$	MSR
0.45	0.1	0.45	0.625812	0.374881	0.682018	0.660000	0.666667
0.4	0.2	0.4	0.674227	0.675892	0.640000	0.560000	1.000000
0.35	0.35	0.3	0.632193	0.599479	0.679393	0.520000	1.000000
0.25	0.45	0.3	0.602516	0.579556	0.684264	0.580000	1.000000
0.2	0.4	0.4	0.631485	0.597244	0.701263	0.580000	1.000000
0.1	0.45	0.45	0.586616	0.357389	0.784097	0.660000	0.888889

According to the above experimental results, when a single failure occurs, all parameter selection cases are able to complete the reconfiguration for that failure. The reconfiguration time share and migration success rate are the same. The algorithm's return value at this point depends on the load balancing situation. From the experiments, we can see that the final choice of the single failure parameter λ_1 Select 0.1.λ_2 Select 0.45.λ_3Choosing 0.45. The impact of load balancing value is more significant than other parameter choices.

When continuous failure occurs, it is not possible to explore a viable reconfiguration option when both 0.45 and 0.1 are selected for λ_1. The metrics for return value are unsatisfactory. When λ_1 non-0.4 is chosen, the return value hardly exceeds 4. According to the experimental results, the return value is more significant than other parameter choices when λ_1 Select 0.4.λ_2 Select 0.2.λ_3 Select 0.4 for continuous failure.

Single Fault Experimental Results. Experiments were primarily conducted on the quality of blueprints and convergence speed.
A.Blueprint quality experiment

The single fault blueprint quality comparison experiment compares four metrics: the return value of the individual reconfiguration algorithm, load balance, reconfiguration time ratio, and migration success rate. The multi-agent algorithm calculates the average value of the agent. To mitigate the instability of reinforcement learning, the data is averaged over 10 rounds of iterations. The single fault has a specific location and less faulty software. The experiment allows for a maximum of 1 fault location in the CPU and a maximum of 3 faulty software. Figure 6 displays the graphs used to compare the quality of reconfiguration blueprints for S1-S4.

According to the figure above, it can be seen that in the case of a single fault, all the various intelligent reconfiguration algorithms have MSR metrics of 1. This means that in the case of fewer software faults, all the algorithms are capable of performing

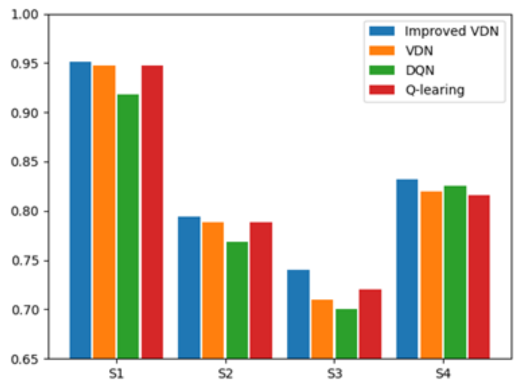

Fig. 6. Comparison of the quality of S1-S4 reconfiguration blueprints

the reconfiguration task. From the data in the table, it can be seen that in the case of the same number of training sessions. The Improved VDN algorithm is capable of generating blueprints with higher load balancing metrics compared to other algorithms. As a result, the reward values also converge to a higher level.
B.Convergence rate experiments

To compare the convergence speed of various reconfiguration blueprint generation algorithms in the presence of a single fault. In this paper, we selected S1 training data generated by an integrated modular avionics system in a typical single fault environment, specifically Fault1, as an example for validation and analysis. The reward values of the four algorithms were compared over 20,000 training rounds. The average reward value per 10 rounds for the total number of training rounds for the Fault1 fault is plotted against the blueprint reward trend, as shown in Figure 7 below. The figure shows the number of converged rounds for the Improved VDN, VDN, DQN, and Q-learning blueprint algorithms based on fault improvement: 665, 1,163, 1,661, and 1,827 rounds, respectively. The Improved VDN algorithm demonstrates a faster convergence speed compared to other algorithms.

Continuous Failure Experiment Results. The following provides a comparison of blueprint quality and convergence speed.
A.Blueprint quality comparison

More faulty software is involved in consecutive faults, with up to 2 CPUs implicated in the fault location and up to 6 instances of faulty software in the experiment. Figure 8 is used to compare the quality of S1-S4 reconfiguration blueprints.

From the figure, it can be seen that there is a continuous failure, such as S6. At this time, the system resources drop dramatically. The corresponding mean squared error (MSR) values of the Improved VDN, VDN, DQN, and Q-learning algorithms are 1, 0.666667, 0.666667, and 0.5, respectively. It shows that in the case of a large number of faulty software, only VDN can generate a usable reconfiguration blueprint within a limited time. The other three algorithms are unable to complete the refactoring and can

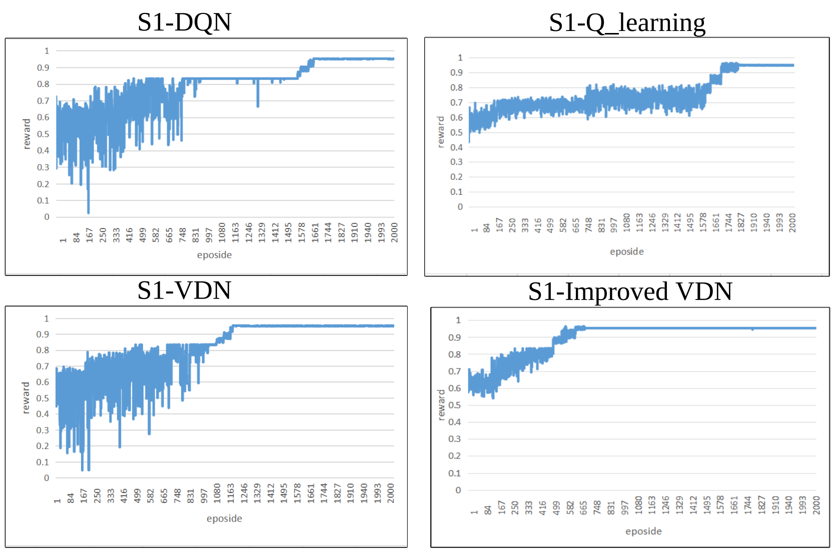

Fig. 7. S1 Blueprint Reward Change Trend Chart

only attempt to migrate the software to be refactored based on the software priority. The software with a lower migration success rate has higher load balancing metrics and refactoring time ratio metrics compared to the Improved VDN algorithm. However, the reward value for final convergence is still lower than that of the Improved VDN algorithm. As can be seen from the data in the table, with the same amount of training, the Improved VDN algorithm is more effective at generating reconfiguration blueprints and converges faster than other algorithms. It also has an advantage over other comparative algorithms in its ability for simultaneous multi-objective optimization.

B.Comparison of convergence speeds

To compare the convergence speed of various reconfiguration blueprint generation algorithms in the presence of continuous faults. In this paper, the training data of the integrated modular avionics system generating S6 under the typical continuous fault environment, Fault6, is selected as an example for validation and analysis. And the algorithms are selected to compare the corresponding reward values of the four algorithms in 20,000 rounds of training. As shown in Table 4, the reward values of the three reconfiguration blueprint generation algorithms changed after 20,000 iterations of training in the presence of Fault1 fault.

For the Fault6 fault, the average reward value of every 10 rounds of training is used to plot the trend of reward change in the blueprint. Under this framework, the exploration factor is adaptively adjusted by utilizing cumulative rewards derived from the average sequence. The number of converged rounds for the improved intelligent reconfiguration blueprint algorithm based on VDN is 1744 rounds. The intelligent reconfiguration blueprint generation algorithms based on VDN, Q-learning, and DQN are all in an oscillating state and cannot converge.

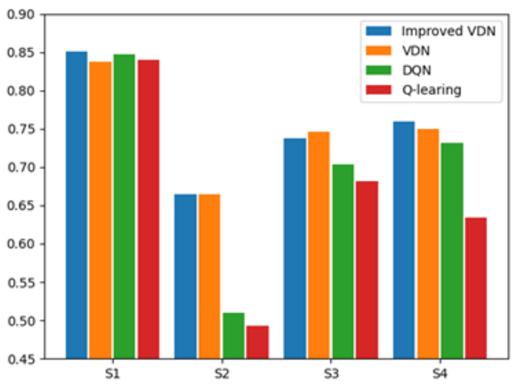

Fig. 8. Comparison of the quality of S5-S8 reconfiguration blueprints

Table 4. S6 State 1-20,000 Training Blueprint Quality Values

Number of iterations	Improved VDN	VDN	DQN	Q-learning
1	0.162586	0.162586	0.137438	0.000125
1000	0.243601	0.243601	0.173120	0.073178
2000	0.321210	0.321210	0.179622	0.180646
3000	0.483040	0.483039	0.192260	0.209591
4000	0.482808	0.482808	0.190402	0.235901
...	...	...	...	...
10000	0.449848	0.449848	0.361703	0.273046
110000	0.432568	0.432568	0.321205	0.291970
120000	0.429175	0.429175	0.396585	0.293272
...	...	...	...	...
18,000	0.670433	0.567463	0.513909	0.493352
19000	0.670433	0.565181	0.515823	0.492240
20000	0.670433	0.569339	0.512242	0.494273

In summary, this paper proposes an intelligent reconfiguration blueprint generation algorithm based on VDN and average sequence cumulative rewards. Compared to traditional linearly decreasing exploration factor strategies, the introduced exploration strategy in this algorithm exhibits higher adaptability and flexibility. It better addresses the learning dynamics of agents during the training process, avoiding blind reduction of the exploration factor that might lead to local optima. Through the design of two reward functions, the system can effectively evaluate the actions of agents, guiding them to achieve a balance between cooperation and competition in multi-agent systems. This enables better completion of the reconfiguration tasks for electronic systems.

A comprehensive analysis of experimental results was conducted, which included parameter selection experiments, single-fault experiments, and continuous-fault experiments. The proposed algorithm demonstrates advantages in both the quality and speed metrics of reconfiguration blueprint generation compared to traditional algorithms based on VDN, DQN, and Q-learning. Furthermore, its advantages become more pronounced, especially in complex scenarios such as continuous faults. The algorithm presented in this paper demonstrates improved applicability and robustness.

Hence, it can be concluded that this paper incorporates the average sequence cumulative reward mechanism into the VDN algorithm and applies it to the comprehensive electronic system reconfiguration blueprint generation method. This approach not only enables the generation of higher-quality reconfiguration blueprints but also achieves the efficient generation of such blueprints.

5. Conclusion

This paper presents the design and validation of an intelligent generation algorithm for reconfiguration blueprints using multi-agent reinforcement learning. The algorithm utilizes average sequential cumulative rewards instead of the conventional VDN algorithm's $\varepsilon\text{-}greedy$ strategy. The algorithm design includes four main elements: integrated modular avionics system state design, exploration and utilization strategy design, action space design, and reward function design. This paper validates the proposed algorithm by comparing its experimental results with reconfiguration blueprint generation algorithms based on traditional VDN, Q-learning, and DQN. The comparison demonstrates that the proposed algorithm exhibits superior convergence characteristics and optimization effects, making it suitable for addressing similar multi-objective optimization reconfiguration problems. In addition, there is room for optimizing load balancing in complex fault environments, and evaluation metrics such as software and module correlation are not considered. In the design of agent systems, as the complexity of faults increases and the number of software components requiring reconfiguration grows, the number of agent systems also increases. This significantly impacts both the solving speed and the convergence rate. Therefore, the definition of an agent system can be further refined. Similarly, bandwidth, ports, routing, topology, and other factors need to be considered in the future to enhance the reconfiguration of distributed integrated electronic systems.

References

1. Hubbard, P.D.: Fault management via dynamic reconfiguration for integrated modular avionics. Computer Science (2015)
2. Chen, J., Du, C., Han, P.: Scheduling independent partitions in integrated modular avionics systems. PLOS ONE 11(12), e0168064 (2016)
3. Wang, P., Zhao, C., Yan, F.: Research on the reliability analysis of the integrated modular avionics system based on the aadl error model. International Journal of Aerospace Engineering (2018)
4. Burger, S., Hummel, O.: Towards automatic reconfiguration of aviation software systems. In: 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops. pp. 200–205. IEEE, Munich, Germany (2011)
5. Burke, M., Audsley, N.: Distributed fault-tolerant avionic systems - a real-time perspective. arXiv - CS - Distributed, Parallel, and Cluster Computing (2010)
6. Wang, H., Niu, W.: A review on key technologies of the distributed integrated modular avionics system. International Journal of Wireless Information Networks 25(12), 358–369 (2018)
7. He, D., Qiao, Q., Gao, J., Chan, S., Zheng, K., Guizani, N.: Simulation design for security testing of integrated modular avionics systems. IEEE Network 34(1), 159–165 (2020)

8. Gui, S., Luo, L., Tang, S., Meng, Y.: Optimal static partition configuration in arinc653 system. Journal of Electronic Science and Technology 9(4) (2011)
9. Blikstad, M., Karlsson, E., Lööw, T., Rönnberg, E.: An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system. Optimization and Engineering 19, 977–1004 (2018)
10. Cui, Y., Shi, J., Wang, Z.: Backward reconfiguration management for modular avionic reconfigurable systems. IEEE Systems Journal 12(1), 137–148 (2018)
11. da Fontoura, A.A., do Nascimento, F.A.M., Nadjm-Tehrani, S., de Freitas, E.P.: Timing assurance of avionic reconfiguration schemes using formal analysis. IEEE Transactions on Aerospace and Electronic Systems 56(1), 95–106 (2020)
12. Suo, D., Zhu, J., An, J.: A new approach to improve safety of reconfiguration in integrated modular avionics. In: 2011 IEEE/AIAA 30th Digital Avionics Systems Conference. pp. 1C4–1–1C4–12. IEEE, Seattle, WA, USA (2011)
13. Huseyinov, I., Bayrakdar, A.: Novel nsga-ii and spea2 algorithms for bi-objective inventory optimization. Studies in Informatics and Control 31(3), 31–42 (2022)
14. He, Z., Li, J., Wu, F., Shi, H., Hwang, K.S.: Derl: Coupling decomposition in action space for reinforcement learning task. IEEE Transactions on Emerging Topics in Computational Intelligence (2023)
15. Housseyni, W., Mosbahi, O., Khalgui, M., Li, Z., Yin, L., Chetto, M.: Multiagent architecture for distributed adaptive scheduling of reconfigurable real-time tasks with energy harvesting constraints. IEEE Access 6, 2068–2084 (2017)
16. Li, J., Shi, H., Hwang, K.: Using goal-conditioned reinforcement learning with deep imitation to control robot arm in flexible flat cable assembly task. IEEE Transactions on Automation Science and Engineering (2023)
17. Shi, H., Li, J., Mao, J., Hwang, K.: Lateral transfer learning for multiagent reinforcement learning. IEEE Transactions on Cybernetics (2021)
18. Jang, B., Kim, M., Harerimana, G., Kim, J.W.: Q-learning algorithms: A comprehensive classification and applications. IEEE access 7, 133653–133667 (2019)
19. Yang, Y., Juntao, L., Lingling, P.: Multi-robot path planning based on a deep reinforcement learning dqn algorithm. CAAI Transactions on Intelligence Technology 5(3), 177–183 (2020)
20. Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J.Z., Tuyls, K., Graepel, T.: Value-decomposition networks for cooperative multi-agent learning. arXiv:1706.05296 (2017)
21. Zhou, Q., Gu, T., Hong, R., Wang, S.: An aadl-based design for dynamic reconfiguration of dima. In: 2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC). pp. 4C1–1–4C1–8. IEEE, East Syracuse, NY (2013)
22. Saadi, A., Oussalah, M., Hammal, Y.: A csp-based approach for managing the dynamic reconfiguration of software architecture. International journal of information technologies and systems approach 14(1), 18 (2021)
23. Chen, L., Wang, L.: Research on functional hazard analysis method for ima reconfiguration. Computer Engineering (2016)
24. Kim, H., Kim, J.: A load balancing scheme for gaming server applying reinforcement learning in iot. Computer Science and Information Systems 17(3), 891–906 (2020)
25. Nguyen, N.T., Luu, L., Vo, P.L., Nguyen, T.T.S., Do, C.T., Nguyen, N.: Reinforcement learning - based adaptation and scheduling methods for multi-source dash. Computer Science and Information Systems 20(1), 157–173 (2023)
26. Zhang, T., Chen, J., Lv, D., Liu, Y., Zhang, W., Ma, C.: Automatic generation of reconfiguration blueprints for ima systems using reinforcement learning. IEEE embedded systems letters 13(4), 182–185 (2021)
27. Liu, B., Zhang, Q., Wang, S.: Automatic generation of reconfiguration blueprints for ima systems using reinforcement learning. In: The 4th Annual IEEE International Conference on Cy-

ber Technology in Automation, Control and Intelligent. pp. 664–668. IEEE, Hong Kong, China (2014)
28. Hollow, P., McDermid, J., Nicholson, M.: Approaches to certification of reconfigurable ima systems. The International Council on Systems pp. 1–8 (2000)
29. Gaska, T., Watkin, C., Chen, Y.: Integrated modular avionics—past, present, and future. IEEE Aerospace and Electronic Systems Magazine 30(9), 12–23 (2015)
30. Balan, N., Ila, V.: A novel biometric key security system with clustering and convolutional neural network for wsn. Tehnički vjesnik 29(5), 1483–1490 (2022)
31. Mani, V., Yarlagadda, S.R., Ravipati, S., Swarnamma, S.C.: Ann optimized hybrid energy management control system for electric vehicles. Studies in Informatics and Control 32(1), 101–110 (2023)
32. Zhang, T., Zhang, W., Dai, L., Chen, J., Wang, L., Wei, Q.: Integrated modular avionics system reconstruction method based on sequential game multi-agent reinforcement learning. ACTA ELECTONICA SINICA 50(4), 954–966 (2022)
33. Wang, H., Zhong, D., Zhao, T.: Avionics system failure analysis and verification based on model checking. Engineering Failure Analysis 105, 373–385 (2019)
34. Li, X., He, D., Gao, Y., Liu, X., Chan, S., Pan, M., Choo, K.: Light: Lightweight authentication for intra embedded integrated electronic systems. IEEE Transactions on Dependable and Secure Computing 20(2), 1088–1103 (2023)
35. Kavitha, S., Uma Maheswari, N., Venkatesh, R.: Intelligent intrusion detection system using enhanced arithmetic optimization algorithm with deep learning model. Tehnički vjesnik 30(4), 1217–1224 (2023)
36. Long, G.L., Shi, L., Xin, G., Gao, S., Zhang, W., Xu, J.: Machine-vision-based online self-optimizing control system for line marking machines. Studies in Informatics and Control 32(2), 93–104 (2023)
37. Liu, Y., Zhang, Y., Jiang, Y., Liu, W., Yang, F.: Uwb-ins fusion positioning based on a two-stage optimization algorithm. Tehnički vjesnik 30(1), 185–190 (2023)
38. dos Santos Mignon, A., de Azevedo da Rocha., R.L.: An adaptive implementation of -greedy in reinforcement learning. Procedia Computer Science 109, 1146–1151 (2017)

Jing Cheng is an associate professor with the School of Computer Science and Engineering, Xi'an Technological University, China. Cheng received a Ph.D. from Northwestern Polytechnical University, China. Her research interests include mobile application testing, crowdsourcing testing, artificial intelligence, and software modeling. Contact her at chengjing@xatu.edu.cn.

Wen Tan is a graduate student at the School of Computer Science and Engineering, Xi'an Technological University. Tan received her bachelor's degree from Xi'an Technological University. Her research interests include reinforcement learning, computer networks. Contact her at tanwen@st.xatu.edu.cn.

Guangzhe Lv is a senior engineer at Xi'an Institute of Aeronautical Technology. He graduated from Xidian University with a master's degree. His research interests include embedded software and distributed systems. Contact him at guangzhe_lv@163.com.

Guodong Li is a researcher at Aviation Industry First Flight Academy and a PhD candidate at Northwestern Polytechnical University. His research interests include aircraft airborne system design, simulation, testing, and experimentation. Contact him at guodonglee@163.com.

Wentao Zhang is a Ph.D. candidate at the School of Software, Northwestern Polytechnical University. He received his master's degree from Northwestern Polytechnical University in 2023. His research interests include reinforcement learning, game optimizationand planning decision making. Contact him at wentaoz1223@gmail.com.

Zihao Liu is a Design Assistant in the Sixth Research Department of Leihua Electronic Technology Research Institute, Aviation Industry Corporation of China. He received his master's degree from Northwestern Polytechnical University. His research interests include embedded, reinforcement learning. Contact him at 2020264414@mail.nwpu.edu.cn.

Received: November 29, 2023; Accepted: June 28, 2024.