Hindawi
Security and Communication Networks
Volume 2022, Article ID 2300407, 23 pages
https://doi.org/10.1155/2022/2300407

Research Article

An Adaptive Multiexecutors Scheduling Algorithm Based on Heterogeneity for Cyberspace Mimic Defense

Zhengbin Zhu,[1] **Hong Yu**,[1,2] **Qinrang Liu**,[1,2] **Dongpei Liu**,[1,2] **and Huapeng Yu**[3]

[1]*Institute of Information Technology, Information Engineering University, Zhengzhou, China*
[2]*National Digital Switching System Engineering & Technological R&D Center, Zhengzhou, China*
[3]*Institute of Cyberspace Security, Zhengzhou University, Zhengzhou, China*

Correspondence should be addressed to Qinrang Liu; ieulqr@163.com

Received 20 June 2022; Revised 1 September 2022; Accepted 5 October 2022; Published 26 December 2022

Academic Editor: Jehad Ali

With the rapid development of network technology, the traditional defense of "mending the fold after the sheep have been stolen" cannot accurately prevent various potential threats and attacks in cyberspace. At the same time, cyberspace mimic defense (CMD) makes the system uncertain and dynamic in time and space to effectively defend against potential attacks. As the key technology of CMD, the scheduling algorithm still needs to be improved in reliability and active defense. Aiming at current problems, this paper first innovatively proposes a new heterogeneous measure algorithm HVTG combined with a vulnerability topology graph, which measures the heterogeneity of executor set in a fine-grained manner. Then, based on the historical confidence, heterogeneity, and minimum sleep time of the executor, we propose an adaptive multi-executors scheduling algorithm (HHAC) to better defend against various attacks. Finally, combining with Analytic Hierarchy Process and Fuzzy Comprehensive Evaluation, this research proposes a comprehensive evaluation model and fill in the gap of the evaluation model of the scheduling algorithm. Theoretical analysis and simulation results show that the HHAC performs well on the system dynamics, probability of system failure, and reliability, which is conducive to the development of CMD.

1. Introduction

With the rapid development of Internet technology, today's world has entered the era of "Internet of everything", and the network has spread to every corner of people's lives. However, there are always uncertain threats in the cyberspace, such as unknown vulnerabilities and backdoors. According to research statistics, programmers will unconsciously produce a vulnerability in every 1000–1500 lines of code [1]. By paying a small price, malicious attackers can infringe on the privacy of individuals and even society, cause large-scale network downtime or failure. In view of the current asymmetric situation in cyberspace, which is easy to attack and difficult to defend, new ideas of active network defense are urgently needed. Wu [2] proposes the theory of cyberspace mimic defense (CMD), which enhances general robustness and endogenous security by introducing dynamic, heterogeneity, redundancy, and closed-loop feedback characteristics into the system [3–6]. The details will be introduced in Section 3.

It is proven by practice that MD greatly increases the difficulty of attacks, what is more, the MD has been successfully applied to routers [7], switches [8], and so on. The realization mechanism of MD mainly includes structural effect and functional fusion, strategy scheduling, mimic ruling, and closed-loop feedback control, which makes the mimic system naturally uncertain both timely and spatially. MD is expected to fundamentally get rid of the current cyberspace dilemma of "Easy to attack, hard to defend."

In Mimic defense, the scheduling algorithm (SA) [9] is the key to realize mimic system high security. It is responsible for building and scheduling online executors according to the historical performance and feedback information and realizing the unpredictability inside the mimic bracket. At the same time, heterogeneity is the vital consideration in the scheduling algorithm, which represents the difference in structure and function between different executors. The greater the heterogeneity, the more difficult to attack two executors at the same time. Furthermore, this article proposes an adaptive multiexecutors scheduling algorithm in CMD, which performs multiexecutors selecting and replacement dynamically according to the changing cyberspace environment. However, there is few research on the scheduling algorithm, and the proposed scheduling algorithm is too random regardless of cost or lack dynamic [10]. Most of the existing scheduling algorithms lack fine-grained research on executor similarity and high-order heterogeneity [11]; some scheduling algorithms mainly depend on feedback mechanism and cannot meet the reliability requirements [12]. In addition, there is no research on SA to analyze the unified evaluation model. Aiming at the shortcomings of existing scheduling algorithms, the main research work in this paper is as follows:

(i) We first summarize the related research on MD and SA. Then, based on an automaton model [13], we illustrate the significance of SA by simulating the state transition process in the mimic system;

(ii) Combining the high-order common vulnerability among executors and graph theory [14], we innovatively propose a new heterogeneous measure algorithm combined with a vulnerability topology graph (HVTG), which can effectively measure the similarity between executors and executor sets;

(iii) Considering the historical confidence of the executor, the high-order heterogeneity of the executor set, and the minimum sleep time of the executor, this paper proposes a multiexecutors adaptive scheduling algorithm (HHAC). The algorithm can adaptively perform online executor behavior transformation according to historical performance and current network environment. Simulation experiments show that HHAC achieves good scheduling overhead and system endogenous security.

(iv) As there is still lacking unified evaluation criteria for SA, we innovatively introduce the Analytic Hierarchy Process (AHP) and Fuzzy Comprehensive Evaluation (FCE) for the comprehensive evaluation of scheduling algorithms. By proposing a general comprehensive evaluation index and model, we finally achieve a good comprehensive evaluation effect.

The structure of this paper is organized as follows: Section 2 introduces the existing research on mimic scheduling algorithms; Section 3 presents a brief overview of MD and SA; in Section 4, a heterogeneous quantification algorithm that considers high-order common vulnerability is proposed; in Section 5, a multiexecutors scheduling algorithm based on high-order heterogeneity and historical confidence is proposed; Section 6 proposes a comprehensive evaluation method based on AHP and FCE; Results are provided and discussed in Section 7; in the last section, the conclusions are drawn, and the scope of future work is discussed.

2. Related Work

As a key part of MD, the SA schedules online executors according to their historical performance and feedback information. It makes the characteristics inside the mimic brackets diverse and unpredictable by rolling the executors online/offline or transferring their services. From the perspective of executor scheduling strategies, this section divides them roughly into three categories: object-based scheduling, time-based scheduling, and quantity-based scheduling. The following will briefly analyze and evaluate them.

2.1. Object-Based Scheduling. Mimic architecture requires online heterogeneous executors have equivalent functions and different structures. The greater the dissimilarity of online executors, the lower the possibility of common vulnerabilities, and the smaller the probability of being successfully attacked via a shared vulnerability. Reference [15] proposes the maximum dissimilarity distance component selection (MD) and the optimal mean dissimilarity distance component selection (OMD) algorithm, which selects the executor set based on the longest dissimilarity and the best average dissimilarity distance, respectively. The system distance threshold setting is relatively high and lacks the dynamic of executors. Lu et al. [16] propose an inverse feedback scheduling algorithm based on historical information to perform optimal scheduling according to different attack types, and the heterogeneity between executors has not been studied. Liu et al. [17] use the similarity between the components of executors to measure their heterogeneity and propose a random seed minimum similarity algorithm (RSMS) to select the executor set with minimum overall similarity, but lack of consideration about executor historical confidence and the dynamic needs further study when the number of executors is small. Zhang et al. [18] take executers' complexity and dissimilarity into consideration, use the quadratic entropy to quantify the dissimilarity of executors, and propose a random seed scheduling algorithm (RSMHQ) based on the maximum heterogeneity and Web service quality, which achieves a better balance between system security and service quality, and it needs to be continuously optimized to determine security and service quality weights according to a different environment. Pu et al. [19] measure the similarity of executors in time and space. Based on common vulnerability between executors, Pu proposes a pool scheduling algorithm based on priority and time slice (PSPT), which achieves good dynamism and time complexity. Reference [20] proposes a random seed scheduling algorithm (RSMHQH) based on executor heterogeneity, performance, and historical confidence, which achieves better performance and comprehensive indexes. Meanwhile, the selection of the seed executor is too random, which will give the attacker a greater chance to attack successfully.

2.2. Time-Based Scheduling. Time-based scheduling is to select an optimal time for online executer set replacement. Lu et al. [21] describe a closed-loop controller scheduling security process and model the time problem of dynamic scheduling as an update process in stochastic theory. At the same time, according to the input scheduling cost, attack loss, and attack distribution function, [21] proposes an optimal scheduling algorithm (OSA) to calculate the optimal scheduling time. The OSA has solved the timing selection of executor to a certain degree and lacks the safe way to select executor and replacement. Considering the abnormal threshold and scheduling period, Guo [22] proposes a scheduling sequence control method based on the sliding window model, which can improve the overall system security through scheduling parameters in different scenarios. The algorithm can improve the system security, high efficiency, and robustness by adjusting the scheduling parameters in different scenarios. However, the algorithm only considers single executor replacement and random selection.

2.3. Quantity-Based Scheduling. On the basis of security, security gain, and cost, Wei et al. [23] comprehensively analyze the three-mode redundancy to obtain the best comprehensive effect between safety and cost. Combining the online redundancy of executor and system dynamics, Qi et al. [24] propose a feedback dynamic-aware scheduling algorithm. According to the number of controller failures to be tolerated by the system, it determines the number of online executors at the next moment. The algorithm reduces the probability of the system at a small cost but needs to reduce complexity. Li [25] proposes a utility-based dynamic elastic scheduling strategy to determine the next online executor redundancy according to the current network environment. Similarly, based on the feedback results and system dynamics, Gao et al. [26] propose a scheduling algorithm to balance cost and security. That is increasing or decreasing the online executor redundancy according to the network environment. But it lacks of the research on the specific transformation time and the update of the subsequent ruling algorithm of executors.

Moreover, in other application fields, Kavin et al. [27] propose a new task scheduling algorithm by combining incremental PSO and A^* Search algorithm in a heterogeneous cloud environment, experiment in the Aneka framework shows that the algorithm is better than existing task scheduling approaches. Gunasekaran [28] proposes a new start up model based on temporal constraints, by scaling back the planning policy for reducing the waiting of scales back tasks and begin times within the Hadoop platform. To optimize the power and performance in public cloud networks, Muthurajkumar et al. [29] propose the Energy Efficient and Optimal Scheduling Algorithm (EEOSA) by applying temporal constraints and rules, finally it has achieved the effective storage and retrieval in cloud data.

To sum up, the SA has specific applications in various fields, and we mainly focus on the application of SA in CMD in this paper. From the aspects of scheduling object, scheduling time and scheduling quantity, the current research results of SA have certain feasibility and application scope and achieve a significant improvement in the security of the mimic system. However, as the key technology of CMD, there is still lacking of the fine-grained and standard metrics for heterogeneity between executors and in-depth research on flexible and adaptive scheduling algorithms. Simultaneously, there is still no standard and effective evaluation for SA. Therefore, taking both system reliability and availability into account, the algorithm is needed to achieve system high-quality security gains without compromising or even improving the original service quality. In view of the existing problems, this paper innovatively proposes a new algorithm HVTG to measure heterogeneous between executors in a fine-grained manner, for example, we consider the high-order common vulnerabilities and vulnerability topology graph that has not been studied. Then, by introducing survival time and historical confidence of executor to adaptive multi-executors scheduling, the HHAC achieves high reliability of online executor set. To the best of our knowledge, we are the first to consider the sleep threshold to select an online executor. Moreover, a comprehensive AHP-FCE model is proposed to evaluate different SAs which can provide a new evaluation criterion to researchers. The above-mentioned contributions will be described in detail below.

3. Related Technology of Mimic Defense

3.1. Mimic Defense Architecture. Based on cyberspace active defense, Wu proposes the theory of MD as shown in Figure 1 [9]. By introducing dynamism, heterogeneity, redundancy, arbitration, and closed-loop feedback mechanisms into the system, the MD system can adjust its internal structure according to the arbitration information while having the inherent uncertainty of the attack surface [30]. Thus, the mimic system can present dynamic and generalized uncertainty to the outside and greatly increases the difficulty of attacks.

As shown in Figure 1, the strategy distribution module of MD distributes k copies of external input to a heterogeneous executor set $E = \{E_1, E_2, \ldots E_k\}$, then E_i executes the task and outputs results to the multimode ruling module. The ruling module chooses the output results according to the ruling algorithm and finally outputs the result. At the same time, the ruling module feeds back the state information of E_i to the scheduling module. Ultimately, the scheduling module decides whether to perform executor scheduling according to the feedback information and current cyberspace environment.

The purpose of SA is to make attackers unable to acknowledge the internal changes of the mimic system through the nonlinear transformation of redundant parts and realize the uncertainty and high security in time and space. This paper summarizes the SA as the following three steps:

(1) According to the feedback information, the SA determines the online executor redundancy and scheduling time, simultaneously selects executors to go online
(2) Replaces the abnormal online executor from time to time, in addition determines the online executor transformation threshold

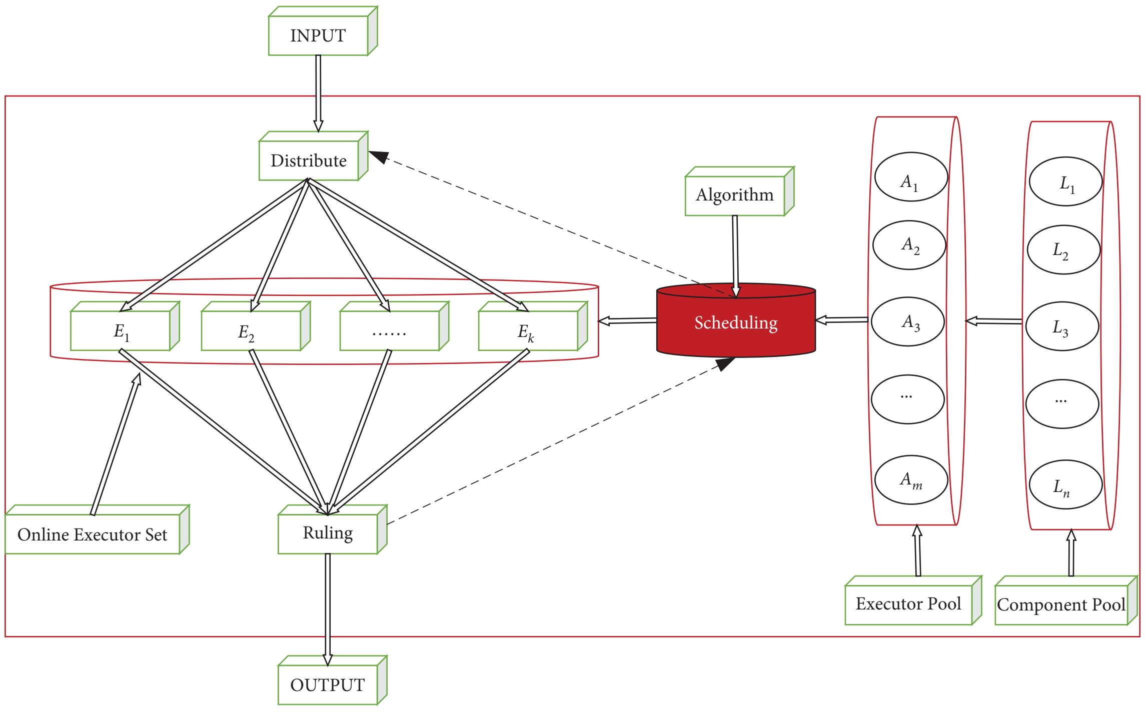


FIGURE 1: Mimic defense architecture.

(3) Outputs the ruling result, and determines the next transformation time and redundancy according to the feedback information

3.2. Importance of SA. In this section, we will use the nondeterministic finite automata model to illustrate the importance of SA in MD. Automata theory is widely used in natural language processing, pattern recognition, automatic control, and other fields. Drawing on the advantages of its powerful and simple state transition function, this section will describe the state transition in MD through automata theory and further explain the importance of SA in MD. Since the online executors can be considered independent and heterogeneous, here we assume that the attacker can only successfully probe and attack a single executor in one scheduling cycle. In addition, there is only one executor can be replaced within a scheduling cycle to ensure high service quality of the system.

In order to briefly describe the role of SA in MD, this section assumes that the online executor redundancy is 3, that is $\{a, b, c\}$. As shown in the state transition in Figure 2, we can see that the state transition is a nondeterministic finite automata model (NFA).

At the same time, the NFA quintuple can be expressed as $M = (Q, \Sigma, \delta, S_0, F)$, which can be expressed as: (Table1).

(1) $\Sigma = \{a, b, c, a_1, b_1, c_1\}$, $Q = \{S_0, S_1, S_2, S_3, S_{12}, S_{13}, S_{23}, S_{123}\}$, among them, $Q_{\text{Instantescape}} = \{S_{12}, S_{13}, S_{23}\}$, $Q_{\text{Permanentescape}} = \{S_{123}\}$, $Q_{\text{normal}} = \{S_0\}$, $Q_{\text{sub-normal}} = \{S_1, S_2, S_3\}$

(2) $\Sigma = \{a, b, c, a_1, b_1, c_1\}$

(3) S_0 is the initial state

(4) $F = \{S_{12}, S_{13,} S_{23}, S_{123}\}$ indicates the attacked success status

(5) Transfer function δ is:

For state $S_i \in \{S_1, S_2, S_3\}$, there are:

$$\begin{cases} \alpha_2 + \alpha_3 = 1 - \eta_1 - \beta, \\ \alpha_1 + \alpha_3 = 1 - \eta_2 - \beta_2, \\ \alpha_1 + \alpha_2 = 1 - \eta_3 - \beta_3. \end{cases} \tag{1}$$

It can be seen from (1) that when $\beta_i\,(1 \le i \le 3)$ is larger, the probability of system state transition to $Q_{\text{Instantescape}}$ is smaller, the system is more secure and difficult to be attacked.

For state $S_j \in \{S_{12}, S_{23}, S_{13}, S_{123}\}$, there are:

$$\begin{cases} \eta_4 + \alpha_3 = 1 - \beta_2 - \beta_1, \\ \eta_5 + \alpha_2 = 1 - \beta_3 - \beta_1, \\ \eta_6 + \alpha_1 = 1 - \beta_2 - \beta_3, \\ \eta_7 = 1 - \beta_2 - \beta_1 - \beta_3. \end{cases} \tag{2}$$

When system state is $S_j \in Q_{\text{Instantescape}}$ or $S_j \in Q_{\text{Permanentescape}}$, the greater the probability $\beta_i\,(1 \le i \le 3)$ at this time, the greater the probability of the system transitioning from attacked successful state to Q_{Normal} or $Q_{\text{Sub-normal}}$. From the NFA above, we can know that the greater the probability SA schedules the wrong executor, the system tends more to return to a normal state and the system is more secure. Through the nonlinear transformation of the mimic system, the attacker cannot grasp the internal changes of the system or attack the bulk of executors successfully. In summary, the SA is crucial in MD.

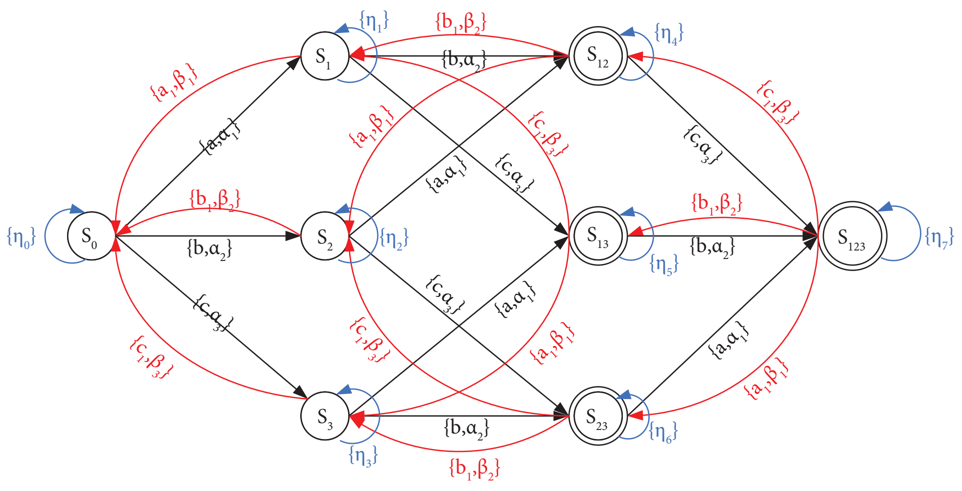


Figure 2: State transition of NFA.

Table 1: State transition function.

δ	(a, α_1)	(b, α_2)	(c, α_3)	(a_1, β_1)	(b_1, β_2)	(c_1, β_3)	η_0	η_1	η_2	η_3	η_4	η_5	η_6	η_7
S_0	S_1	S_2	S_3	∅	∅	∅	S_0	∅	∅	∅	∅	∅	∅	∅
S_1	∅	S_{12}	S_{13}	S_0	∅	∅	∅	S_1	∅	∅	∅	∅	∅	∅
S_2	S_{12}	∅	S_{23}	∅	S_0	∅	∅	∅	S_2	∅	∅	∅	∅	∅
S_3	S_{13}	S_{23}	∅	∅	∅	S_0	∅	∅	∅	S_3	∅	∅	∅	∅
S_{12}	∅	∅	S_{123}	S_2	S_1	∅	∅	∅	∅	∅	S_{12}	∅	∅	∅
S_{13}	∅	S_{123}	∅	S_3	∅	S_1	∅	∅	∅	∅	∅	S_{13}	∅	∅
S_{23}	S_{123}	∅	∅	∅	S_3	S_2	∅	∅	∅	∅	∅	∅	S_{23}	∅
S_{123}	∅	∅	∅	S_{23}	S_{13}	S_{12}	∅	∅	∅	∅	∅	∅	∅	S_{123}

4. Heterogeneous Measure

4.1. Threat of High-Order Common Vulnerability. Existing scheduling algorithms mostly calculate the heterogeneity between two executors by their common vulnerabilities and calculate the heterogeneity of executor set by accumulating the heterogeneity between each two executors, which lacks consideration of high-order common vulnerabilities. Zhang [31] proposes the concept of high-order common vulnerability as:

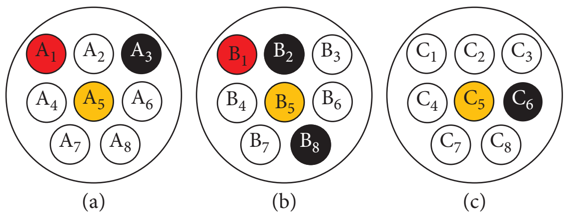


Figure 3: High-order vulnerability.

Definition 1. (High-order Common Vulnerabilities) When there are vulnerabilities in different executors that can achieve the same attack effect, and the number of executors meeting this situation is m, then we define them as m-order common. In addition, when $m \geq 3$, we call them high-order common vulnerabilities.

As shown in Figure 3, assuming that the online executor redundancy is 3, then there is $E = \{A, B, C\}$. $E_i \in E$ can be expressed as $E_i = \{e_1, e_2, \ldots\ldots e_8\}$, in which e_i stands for a component of E_i. When there is a vulnerability that exists in the component e_i of A, as well as B, and C, and it can cause the same attack effect, then component e_i in A, B, and C are all marked. We use $\mathrm{VUL}_i = \{v_1, v_2, \ldots\ldots v_8\}$ to indicate all the vulnerabilities existing in the executer set. If $\mathrm{VUL}_i = 0$, then there is no vulnerability in A, Band C. For example, we can tell that v_5 is a 3-order common vulnerability of executor set $E = \{A, B, C\}$. Similarly, v_1 is a 2-order common vulnerability, and v_2, v_3, v_6, v_8 are1-order common vulnerabilities.

We assume that an attacker can only discover and exploit one vulnerability in one scheduling cycle. As shown in Figure 4(a), attacker discovers and exploits vulnerability v_6, it can be used to attack executor C successfully, and C only. As executors A and B are not attacked successfully, the system can still output correct results after the majority ruling. If the attacker discovers and exploits the 2-order common vulnerability v_1, as shown in Figure 4(b), the executors A and B are successfully breached at the same time. Then, the final ruling result is wrong and system will be breached momentarily (we call this phenomenon an instant attack escape). Moreover, there is 3-order common vulnerability v_5 in the executor set. As shown in Figure 4(c), if the attacker exploits vulnerability v_5 and attacks all online executors successfully, the system will be breached and the

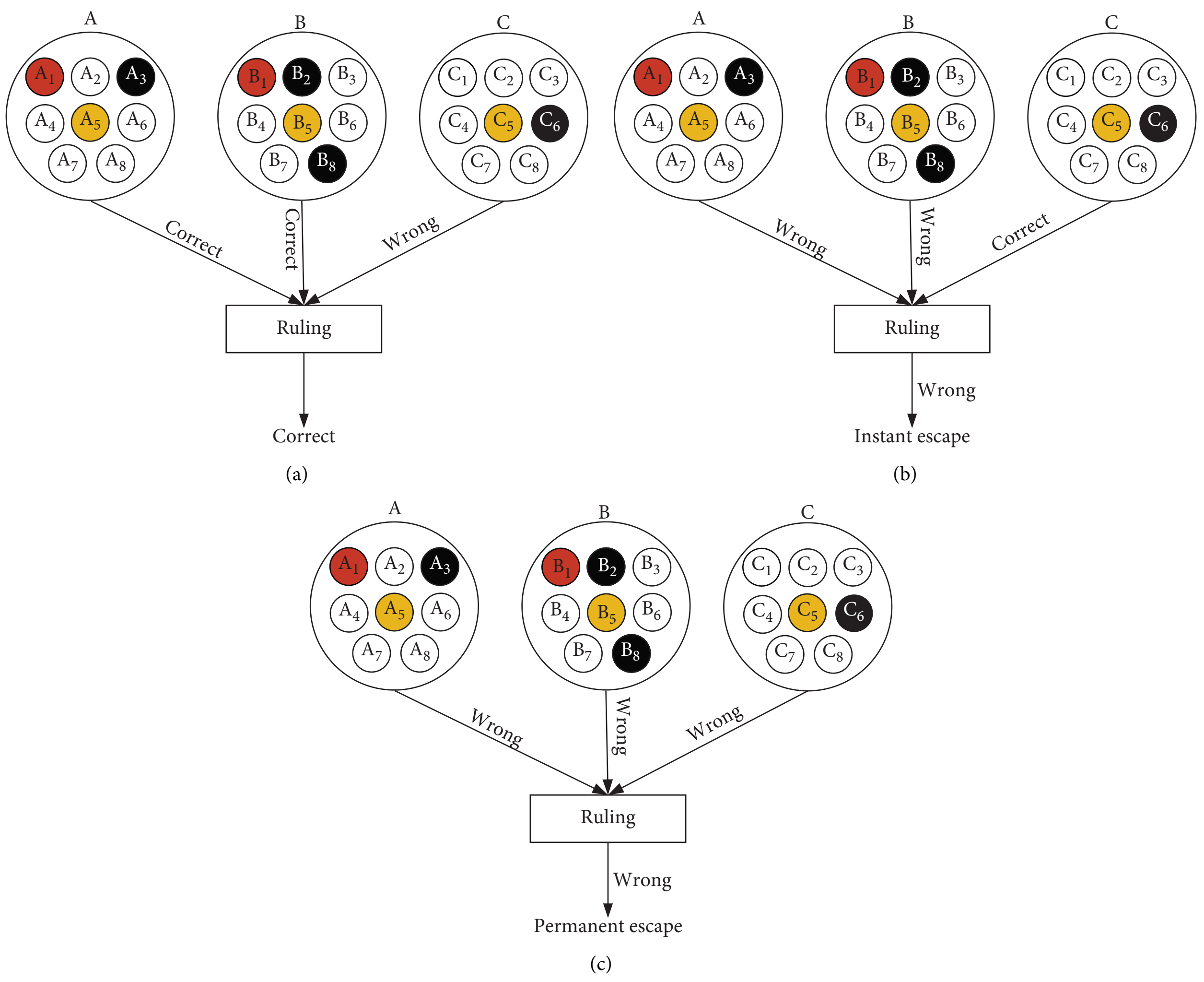


FIGURE 4: Threat of high-order vulnerability.

inverse feedback control mechanism will not be triggered until the next scheduling cycle. To sum up, the existence of high-order common vulnerabilities in the executor set will seriously threaten the security of the mimic system.

4.2. Heterogeneous Measure. Most of the existing heterogeneity measuring algorithms between two executors are based on their common vulnerabilities (which is also the 2-order common vulnerabilities of these two executors). After the analysis above, this paper concludes that the existing methods for calculating heterogeneity of the executor set still have the following deficiencies:

(1) Only the similarity of the same component between different executors (based on common vulnerabilities between two executors) is considered, and there may be common vulnerabilities between different components of the same or different executors
(2) The complexity of matrix operation, which is based on executor components, turns too high when there are too many executors in the executor pool
(3) There is no consideration for the existence or calculation method of high-order common vulnerabilities

Aiming at the deficiencies above, taking both high-order common vulnerabilities and computational complexity into consideration, this section creatively proposes the heterogeneous measure algorithm based on a vulnerability topology graph (HVTG). Using graph theory, we construct the vulnerability topology graph based on the vulnerabilities that have been found between executors. Assuming that the online executor set is $E = \{E_1, E_2, \dots E_m\}$, and each executor is composed of n components, that is $\forall E_i \in E$, $E_i = \{C_i^1, C_i^2, \dots C_i^n\}\,(1 \le i \le m)$. Then the online executor set of the system can be expressed as follows:

$$E = \begin{pmatrix} C_1^1 & C_1^2 & \cdots & C_1^n \\ C_2^1 & C_2^2 & \cdots & C_2^n \\ \vdots & \vdots & \ddots & \vdots \\ C_m^1 & C_m^2 & \cdots & C_m^n \end{pmatrix}. \quad (3)$$

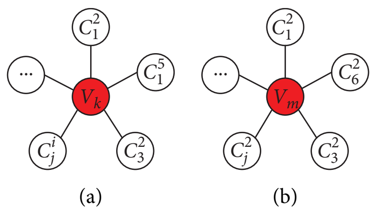


Figure 5: Vulnerability topology graph.

Among them, each row of matrix E, that is $(C_i^1, C_i^2, \dots C_i^n)$, represents an executor, and each column, that is $(C_1^j, C_2^j, \dots C_m^j)$, represents a class of functionally equivalent components.

Definition 2. (Vulnerability Topology Graph (VTG)) VTG is constructed based on the discovered vulnerabilities in all executors. Suppose the discovered vulnerabilities are recorded as $V = \{v_1, v_2, \dots v_M\}$, for $\forall v_k \in V$, its component set can be represented as $C_k = \left\{C_i^j\right\} i \in (1, m), j \in (1, n)$. As shown in Figure 5, the VTG is formed by connecting the vulnerability v_k with its component set.

As we can see from Figure 5, the VTG takes multiple situations into consideration. Firstly, the situation in which common vulnerabilities appear in the same component of different executors. For example, in Figure 5(b), there is the same vulnerability v_m in the component C_i^2 of executor E_1, E_3, and E_6. Secondly, the situation in which common vulnerabilities appear in different components of different executors. As shown in Figure 5(a), the component C_1^5 of executor E_1 shares a vulnerability v_k with the components C_3^2 of executor E_3 . Thirdly, the situation in which common vulnerabilities appear in different components of the same executor, such as the components C_1^5 and C_1^2 of E_1 share a common vulnerability v_k. In summary, the VTG can calculate the location of vulnerability and select the online executor set in a more fine-grained manner.

Definition 3. (Pseudo n-order common vulnerability) As shown in Figure 5(a), when vulnerability v_k has n branches, that is, $|C_{\text{reality}}^k| = n$.But within its component set, exists a component subset of the same executer, for example $(C_i^{j_1}, C_i^{j_2}, \dots C_i^{j_n})$.For $\forall v_k \in V, E_i \in E$, there is $C_i^{j_1} \in C_k, C_i^{j_2} \in C_k, \dots C_i^{j_n} \in C_k$. At this time, we define v_k as a pseudo n-order common vulnerability.

Definition 4. (n-order common vulnerability) $\forall v_k \in V$, the component set can be expressed as $C_k = \left\{C_i^j, i \in (1, m), j \in (1, n)\right\}$. Calculate its maximum subset, components in the same executor are only calculated once. The maximum subset $C_{\max}^k \subseteq C_k$ satisfies $C_{\max}^k = \left\{C_{i_1}^j, C_{i_2}^j, \dots C_{i_n}^j, j \in (1, n)\right\}$, when $|C_{\max}| = n$, we define vulnerability v_k as n-order common vulnerability.

The pseudo n-order common vulnerability increases the attack surface of executor E_i for vulnerability v_k. Here we have reason to believe that the threat of pseudo n-order common vulnerability is greater than $C_{\max}^k$ of vulnerability v_k, but less than n-order common vulnerability. To elaborate on n-order common vulnerability, we assume an executor set $E = \{E_1, E_2, E_3, E_4, E_5\}$, for $\forall E_i \in E$, $E_i = \{C_i^1, C_i^2, C_i^3, C_i^4, C_i^5\}\,(1 \le i \le 5)$. Assuming that vulnerability set $V = \{v_1, v_2, v_3, v_4, v_5\}$ has been found in E, the VTG is shown in Figure 6.

From Figure 6, we can know that v_1 is a 5-order common vulnerability, and v_2 is a pseudo 5-order common vulnerability but actually a 4-order common vulnerability. The same, v_3 is a pseudo 3-order vulnerability, v_4 is a 3-order common vulnerability, and v_5 is a 2-order common vulnerability. Considering the VTG above, this section calculates the threat level of n-order common vulnerability v_k from structure. The higher the vulnerability order, the higher the system similarity, and the greater the potential threat of the system. Taking the pseudo n-order common vulnerability and n-order common vulnerability into consideration, here we define the weight function of v_k as follows:

$$\alpha(x, y) = \frac{1}{1 + e^{-(x + 1/2y - n + 1/2)}}, \tag{4}$$

where x represents the order of vulnerability v_k, y represents the bias order calculated as $y = |C_{\text{reality}}^k| - |C_{\max}^k|$, and n represents the online executor redundancy.

Proof. The calculation of n-order common vulnerability weight $\alpha(x, y)$ should conform to the change rule of vulnerability threat degree. Firstly, $\alpha(x, y)$ must be a monotonically increasing function. Then, the vulnerability threat degree should increase faster when $x \in [n - 1/2, n + 1/2]$ or $y \in [n - 1/2, n + 1/2]$. First, we calculate the first partial derivative of function α with respect to variable x:

$$\frac{\partial \alpha}{\partial x} = \frac{e^{-(x + 1/2y - n + 1/2)}}{\left(1 + e^{-(x + 1/2y - n + 1/2)}\right)^2} 0. \tag{5}$$

Let $y \in C$, (5) is equivalent to:

$$\frac{\partial \alpha}{\partial x} = \frac{e^{-(x - n + 1/2)}}{\left(1 + e^{-(x - n + 1/2)}\right)^2}. \tag{6}$$

Then, calculate the second partial derivative of x:

$$\begin{aligned}\frac{\partial^2\alpha}{\partial x^2} &= \frac{-e^{-(x-n+1/2)}\left(1+e^{-(x-n+1/2)}\right)^2 + 2e^{-(x-n+1/2)}\left(1+e^{-(x-n+1/2)}\right)e^{-(x-n+1/2)}}{\left(1+e^{-(x-n+1/2)}\right)^4} \\ &= \frac{e^{-(x-n+1/2)}\left(e^{-(x-n+1/2)}-1\right)}{\left(1+e^{-(x-n+1/2)}\right)^3}.\end{aligned} \tag{7}$$

In the same way, calculate the second partial derivative of function α with respect to variable y when $x \in C$:

$$\begin{aligned}\frac{\partial^2\alpha}{\partial y^2} &= \frac{-1/2e^{-(1/2y-n+1/2)}\left(1+e^{-(1/2y-n+1/2)}\right)^2 + e^{-(1/2y-n+1/2)}\left(1+e^{-(1/2y-n+1/2)}\right)e^{-(1/2y-n+1/2)}}{\left(1+e^{-(1/2y-n+1/2)}\right)^4} \\ &= \frac{e^{-(1/2y-n+1/2)}\left(e^{-(1/2y-n+1/2)}-1\right)}{2\left(1+e^{-(x-n+1/2)}\right)^3}.\end{aligned} \tag{8}$$

Finally, calculate the second partial derivative of x:

$$\frac{\partial^2\alpha}{\partial x^2} = \begin{cases} >0 & x \in \left[1, \dfrac{n+1}{2}\right), \\ =0 & x = \dfrac{n+1}{2}, \\ 0 & x \in \left(\dfrac{n+1}{2}, n\right]. \end{cases} \tag{9}$$

From (6)–(8), we can conclude that $\partial\alpha/\partial x 0$, that is weight function $\alpha(x, y)$ increases monotonically. Furthermore, the value of $\partial\alpha/\partial x$ reaches maximum when $x = (n+1)/2$, that is, the vulnerability weight increases the fastest when $x = (n+1)/2$, which is in line with the fact that when the vulnerability order is more than half of the number of online executors, the threat degree of vulnerability will increase sharply and cause instant escape. At the time when $x \in ((n+1)/2, n]$, the vulnerability weight gradually increases but its increasing speed gradually decreases, which satisfies the change law of vulnerability threat degree. □

Definition 5. (Vulnerability Binary Subset (VBS)) For $v_k \in V$, the vulnerability component set can be expressed as $C_k = \{C_i^j, i \in (1, m), j \in (1, n)\}$. Based on C_k, we define VBS to contain two elements and $C_2^k \subset C_k$. The number of VBS of vulnerability v_k is $|C_2^k| = C_{|C_k|}^2$. For executors E_i, E_j, let:

$$z_q = \begin{cases} 1 & C_j^l \text{and } C_i^o \text{are in } C_2^k \text{simultaneously}, l, o \in (1, n), \\ 0 & C_j^l \text{and } C_i^o \text{are not in } C_2^k \text{simultaneously}, l, o \in (1, n). \end{cases} \tag{10}$$

Traversing the VBS of vulnerability v_k, then the heterogeneity between executors E_i, E_j can be expressed as follows:

$$H_{E_iE_j} = \sum_{k=1}^{M}\left(\alpha_k \sum_{q=1}^{C_{|C_k|}^2} z_q\right). \tag{11}$$

From (4)–(10), we can conclude that the larger $H_{E_iE_j}$ is, the more common vulnerabilities there are between E_i, E_j, the greater the possibility of high-order common vulnerabilities, and the higher the similarity between E_i, E_j. When $H_{E_iE_j} = 0$, there is complete heterogeneity between E_i, E_j. On the contrary, executors E_i, E_j are exactly the same when $H_{E_iE_j} = 1$. The similarity of online executor set in HTVG can be calculated as follows:

$$H_E = \sqrt[m]{\sum_{i=1}^{m-1}\sum_{j=i+1}^{m} H_{E_iE_j}^m}. \tag{12}$$

Finally, the pseudocode of HTVG is shown in Algorithm , Lines 1–9 construct the VTG and VBS by all discovered vulnerabilities in executors. From lines 10 to 17, we determine the order and weight of each vulnerability according to the function Get-order-vulnerability. In lines 18–22, we compute the heterogeneity of executors and executor set. In summary, this paper achieves the measure of similarity between executors in a more fine-grained way and avoids the occurrence of high-order vulnerability to the greatest extent.

5. Scheduling Algorithm Based on High-Order Heterogeneity and Adaptive Historical Confidence (HHAC)

5.1. The Measure of Historical Confidence. Measure the historical performance of the executor in the past to get its historical confidence of executor (HCE), which can reflect its historical performance and current ability to resist attack. Most of the current research studies calculate the global

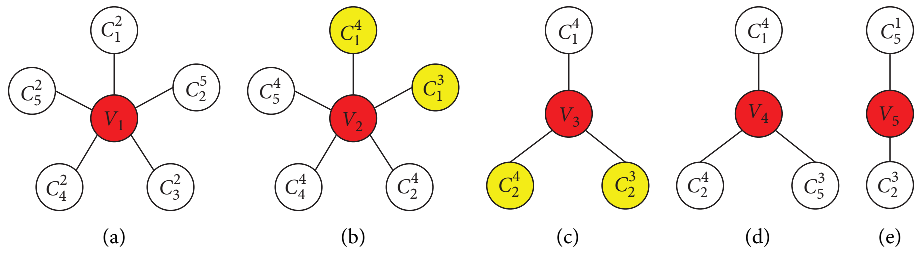


Figure 6: Example of VTG.

```
**Assumption:** The relationship of vulnerabilities and components between executors have been found;
**Input:** The relationship of vulnerabilities and components between executors
**Output:** Heterogeneity of executor set $H_E$
(1) / * **Construct VTG** * /
(2) For $v_k \in V$
(3) $k = 1$
(4) $v_1 \leftarrow C_1 = \{C_i^j\} i \in (1, m), j \in (1, n)$
(5) $k++$
(6) if $kM$, then
(7) Endif
(8) $\text{Graph}_E \leftarrow [v_1, v_2, \ldots v_M]$
(9) $\text{Graph}_E \leftarrow [E]$
(10) / * Determine order and weight of vulnerability * /
(11) Get-order-vulnerability $[v_1, v_2, \ldots v_M]$
(12) For $v_k \in V$
(13) $k = 1$
(14) $\text{Order}_1 \leftarrow \text{Get} - \text{order} - \text{vulnerability}(v_k)$
(15) $\alpha_k \leftarrow \alpha_k(x, y)$
(16) $k++$
(17) if $kM$ end
(18) / * Compute heterogeneity of two executors * /
(19) $H_{E_iE_j} \leftarrow \sum_{k=1}^{M} \alpha_k z_i$
(20) / * Compute heterogeneity of executor set * /
(21) $H_E \leftarrow \sqrt[m]{\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} H_{E_iE_j}^m}$
(22) Return $H_E$, $H_{E_iE_j}$.
**Function Get-order-vulnerability**. $[v_1, v_2, \ldots v_M]$
(1) For $v_k \in V$
(2) $C_{\max}^k \subseteq C_k$, Satisfy $C_{\max}^k = \{C_{i_1}^j, C_{i_2}^j, \ldots C_{i_n}^j, j \in (1, n)\}$
(3) if there is no $C_i^{j_1} \in C_k, C_i^{j_2} \in C_k, \ldots C_i^{j_n} \in C_k$
(4) $\text{Order}_k \leftarrow |C_{\max}^k|$
(5) else
(6) $x \leftarrow |C_{\max}^k|$
(7) $y \leftarrow |C_{\text{reality}}^k| - |C_{\max}^k|$
(8) endif
```

Algorithm 1: Heterogeneous measure algorithm based on VTG (HTVG).

confidence [22], namely, the whole historical performance of the executor. Furthermore, Zhang [28] proposes the sliding window confidence by calculating the HCE of the current local time period. However, the global confidence of an executor cannot completely reflect its real attacked state in the current time period, and the sliding window confidence only considers the attacked state in the current time period. From what has been discussed above, we believe that both the global and local historical confidence of the executor should be considered, and this paper redefines the concepts and calculation methods of both.

Definition 6. (Global Confidence (GC)) As shown in Figure 7, we express the time points when executor E_i rolls online as $t_{\text{on}} = \{t_0, t_n, t_j\}$, and the time points when executor E_i rolls offline as $t_{\text{off}} = \{t_m, t_k\}$. This paper defines the historical performance of executor E_i during $[t_0, t_j]$ as the global confidence $\text{CON}_{\text{global}}$.

Definition 7. (Local Confidence (LC)) As shown in Figure 7, $[t_j, t_{j+i}]$ represents the time period after executor E_i rolled online at t_j., and E_i may be replaced offline or still performing online at t_{i+j}. From above, we define the recent performance of executor E_i during $[t_j, t_{j+i}]$ as the local confidence $\mathrm{CON}_{\mathrm{local}}$.

This section proposes an adaptive measure algorithm for HCE based on executer's the online working time and number of performed tasks, and then introduces the calculation method of $\mathrm{CON}_{\mathrm{global}}$ and $\mathrm{CON}_{\mathrm{local}}$ in detail. Relevant parameters are shown in Table 2.

According to Definition 6, we calculate $\mathrm{CON}_{\mathrm{global}}$ using the formula as follows:

$$\mathrm{CON}_{\mathrm{global}} = \frac{T^{E_i}_{\mathrm{on\_time}} + N^{E_i}_{\mathrm{on\_task}}}{T_{\mathrm{time}} + N_{\mathrm{task}}}. \tag{13}$$

As for $\mathrm{CON}_{\mathrm{local}}$, since executor E_i still works online during $[t_j, t_{j+i}]$, we perform an adaptive update after each task, and the update rule should be in consistency with the change rule of historical confidence with task success or failure. The formula is

$$\mathrm{CON}^{j+i}_{\mathrm{local}} = \mathrm{CON}^{j}_{\mathrm{local}} + \frac{\mathrm{d}(\mathrm{CON}_{\mathrm{local}})}{\mathrm{d}t}. \tag{14}$$

Among them,

$$\mathrm{CON}^{j}_{\mathrm{local}} = \begin{cases} 0.5 & \mathrm{E}_i \text{ firstonline}, \\ \dfrac{T^{E_i}_{\mathrm{on\_time}} + N^{E_i}_{\mathrm{on\_task}}}{T_{\mathrm{time}} + N_{\mathrm{task}}} & \mathrm{E}_i \text{ onlineagain}. \end{cases} \tag{15}$$

When E_i performs a task successfully during $[t_j, t_{j+i}]$, the LC of E_i should increase slowly. On the contrary, the LC should decrease rapidly. In addition, if the number of wrong outputs reaches the threshold, E_i must be reduced to the threshold value of offline cleaning. If E_i performs a task successfully at t_{j+i}, then

$$\frac{\mathrm{d}(\mathrm{CON}_{\mathrm{local}})}{\mathrm{d}t} = \frac{T^{*}_{\mathrm{time}} + N^{*}_{\mathrm{task}} - \left(T^{i^*}_{\mathrm{on\_time}} + N^{i^*}_{\mathrm{on\_task}}\right) + 2}{\left(T^{*}_{\mathrm{time}} + N^{*}_{\mathrm{task}}\right)\left(T^{*}_{\mathrm{time}} + N^{*}_{\mathrm{task}} + 2\right)}. \tag{16}$$

If E_i performs a task unsuccessfully at t_{j+i} (moreover, this error output is the *kth* time during $[t_j, t_{j+i}]$), then

$$\frac{\mathrm{d}(\mathrm{CON}_{\mathrm{local}})}{\mathrm{d}t} = -\frac{(k+1)^2\left(T^{i^*}_{\mathrm{on\_time}} + N^{i^*}_{\mathrm{on\_task}} + 1\right)}{T^{*}_{\mathrm{time}} + N^{*}_{\mathrm{task}} + 2}. \tag{17}$$

Proof. After performing a task successfully, let $n = T^{*}_{\mathrm{time}} + N^{*}_{\mathrm{task}}, m = T^{i^*}_{\mathrm{on\_time}} + N^{i^*}_{\mathrm{on\_task}} (nm)$, then the growth rate of LA is

$$\frac{\mathrm{d}(\mathrm{CON}_{\mathrm{local}})}{\mathrm{d}t} = \frac{n-m+2}{n(n+2)}\frac{1}{n} - \frac{m}{n(n+2)} < \frac{1}{n}. \tag{18}$$

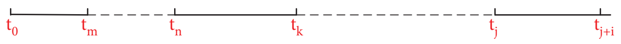


FIGURE 7: Task time period of E_i.

Similarly, when E_i performs a task unsuccessfully, the change rate of LC can be calculated as follows:

$$\frac{\mathrm{d}(\mathrm{CON}_{\mathrm{local}})}{\mathrm{d}t} = -\frac{(k+1)^2\left(T^{i^*}_{\mathrm{on\_time}} + N^{i^*}_{\mathrm{on\_task}} + 1\right)}{T^{*}_{\mathrm{time}} + N^{*}_{\mathrm{task}} + 2} > -(n+1)^2. \tag{19}$$

According to (16)–(18), the confidence of E_i increases slowly by about $1/(T^{*}_{\mathrm{time}} + N^{*}_{\mathrm{task}})$ after performing a task successfully, and decrease rapidly by about $(T^{*}_{\mathrm{time}} + N^{*}_{\mathrm{task}} + 1)^2$ after outputting a wrong result. That is, when the number of wrong output results increases, the decline amplitude of $\mathrm{CON}^{E_i}_{\mathrm{local}}$ gradually increases. To sum up, we calculate the GC to measure the historical performance of $E_i$and use $\mathrm{CON}_{\mathrm{global}}$ as input to calculate $\mathrm{CON}_{\mathrm{local}}$. Furthermore, $\mathrm{CON}_{\mathrm{local}}$ is adjusted adaptively according to the efficiency of the current task, so it can better quantify E_i's current ability to resist attack risk. In order to avoid frequent executor scheduling due to occasional output errors or inconsistencies, we propose the adaptive measure algorithm for HCE to provide priori information and the trust degree of the executor for SA. □

5.2. The Selection of Adaptive Survival Time. Assuming the attacker is smart enough, and the attacker may discover a vulnerability but do not attack immediately in this scheduling cycle, instead, wait until obtaining the vulnerability of another online executor or vulnerabilities of more than half of the online executors. Then, the attacker attacks by exploiting all the vulnerabilities at the same time, which can cause instant attack escape or permanent attack escape due to more than half of executors output wrongly. Considering the above-given potential threats, this paper introduces the survival time disturbance factor. E_j carries a random survival time, and E_j will go offline regardless of its historical performance or current potential threats when the online time of E_j exceeds its random survival time. The executor scheduling process with survival time disturbance factor is shown in Figure 8. According to the rotation scheduling mentioned earlier, the optimal backup executor $B_i \in B$ is calculated for each online executor E_j to form the backup executor set $B = \{B_1, B_2, \ldots B_k\}$. When the historical confidence of E_j is lower than the threshold or the survival time is reached, B_i will be scheduled online and given a survival time.

Definition 8. (Random Survival Time Disturbance Factor ϕ_{random}) To avoid attacks when executor E_j is initially online, we introduce a random survival time disturbance factor ϕ_{random} to determine its online survival time regardless of its historical confidence or current performance.

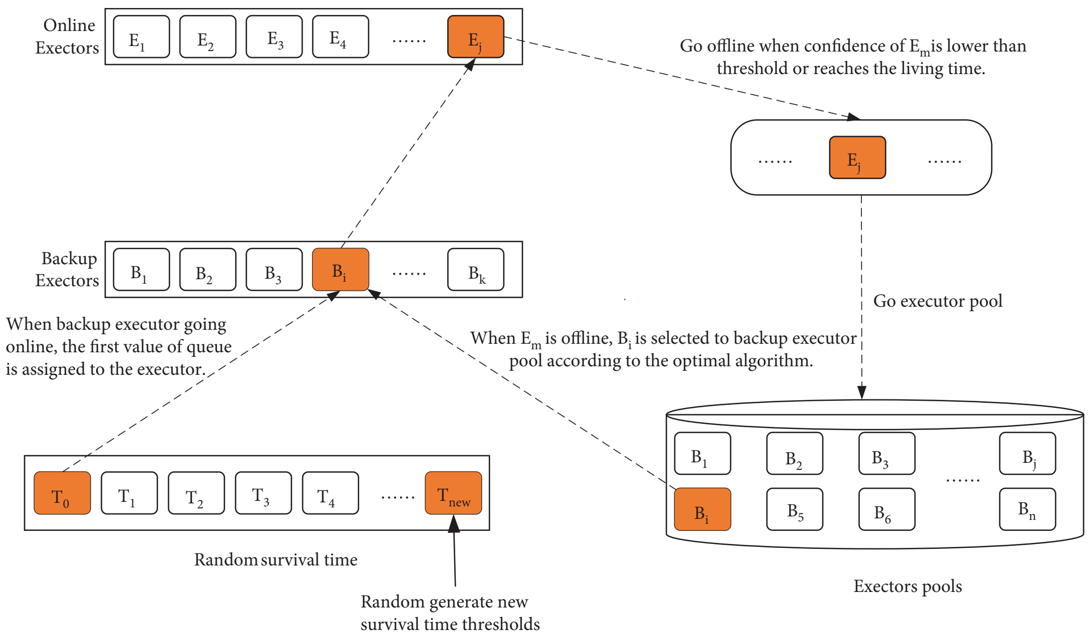

Online Exectors
E1
E2
E3
E4
......
Ej
Go offline when confidence of Em is lower than threshold or reaches the living time.
...... Ej ......
Backup Exectors
B1
B2
B3
Bi
......
Bk
Go executor pool
When backup executor going online, the first value of queue is assigned to the executor.
When Em is offline, Bi is selected to backup executor pool according to the optimal algorithm.
T0
T1
T2
T3
T4
......
Tnew
B1 B2 B3 ...... Bj
Bi B5 B6 Bn
Random survival time
Random generate new survival time thresholds
Exectors pools

FIGURE 8: Dynamic scheduling model of survival time.

Definition 9. (Adaptive Survival Time Disturbance Factor ϕ_{adaptive}) An overly short survival time can cause frequent replacement of executers and furthermore high system cost. While an overly long survival time can bring relatively high threat potentials. In order to avoid random survival time getting to short or too long, we should adaptively increase or decrease the online time according to the historical confidence of E_j, and we introduce the adaptive survival time disturbance factor ϕ_{adaptive} to do so.

In this section, we propose an adaptive generation iterative algorithm of the survival time disturbance factor ϕ_{adaptive} based on $\text{CON}_{\text{local}}$. Considering the LC of E_i, and the calculation method of ϕ_{random} and ϕ_{adaptive} will be introduced in detail below.

(1) Regardless of adaptive change of survival time, when executor E_j is initially online at t_0 in a scheduling cycle, the random survival time disturbance factor ϕ_j^0 will be generated, that is, E_j will go offline into cleaning at time $t_{\text{off}} = t_0 + \phi_j^0$;

(2) When $t_i \in [t_0, t_{off} + +]$, the adaptive transformation rule is

$$\phi_j^{t_i} = \begin{cases} \phi_j^0 - \int_0^{t_i} 2t/1 + t^2 dt, & \text{CON}_{th}\text{CON}_{\text{local}}^{t_i} \leq 2\text{CON}_{th}, \\ \phi_j^0 - \int_0^{t_i} 1/1 + t dt, & \text{CON}_{\text{local}}^{t_i} > 2\text{CON}_{th}. \end{cases} \tag{20}$$

TABLE 2: Notations.

Notations	Definition
$\text{CON}_{\text{global}}$	Global confidence of executor
$\text{CON}_{\text{local}}$	Local confidence of executor
T_{time}	Running time of system
N_{task}	Total number of tasks of system
$T^{E_i}_{\text{on\_time}}$	Total online time of E_i at this moment
$N^{E_i}_{\text{on\_task}}$	Total number of tasks of E_i at this moment
T^*_{time}	Last total online time of E_i
N^*_{task}	Last total number of tasks of system between $[t_j, t_{j+i}]$
$T^{i*}_{\text{on\_time}}$	Total time last online of E_i
$N^{i*}_{\text{on\_task}}$	Total number of tasks of E_i between $[t_j, t_{j+i}]$

5.3. Scheduling Algorithm Based on High-Order Heterogeneity and Confidence (HHAC). When selecting an initial online executor set, we tend to think that the ideal scheme is to select a set that meets the redundancy condition and has the overall minimum similarity. But the fact may prove otherwise, that the set with overall minimum similarity may not be the optimal choice. To explain the problem, here we consider a 3-redundant mimic defense system as is shown in Figure 9(a), the executor pool is $E = \{A_1, A_2, A_3, A_4\}$, and $L_{ij}\,(1 \leq i, j \leq 4)$ represents heterogeneity between executor E_i and E_j. Then, there are two scheduling schemes $S_1 = \{A_1, A_2, A_3\}$ and $S_2 = \{A_1, A_3, A_4\}$, because $L_{12} + L_{23} + L_{12}L_{14} + L_{34} + L_{13}$, namely the overall heterogeneity of S_2 is greater than S_1. But in S_2, $L_{34}L_{ij}, L_{ij} \in \{L_{12}, L_{23}, L_{12}\}$, that is, the probability of common vulnerabilities between A_3 and A_4 is relatively high, and the probability of attackers successfully exploiting the common vulnerabilities is relatively high. Therefore, when using the similarity index, we should not only consider the overall value but also prevent the occurrence of local extreme values.

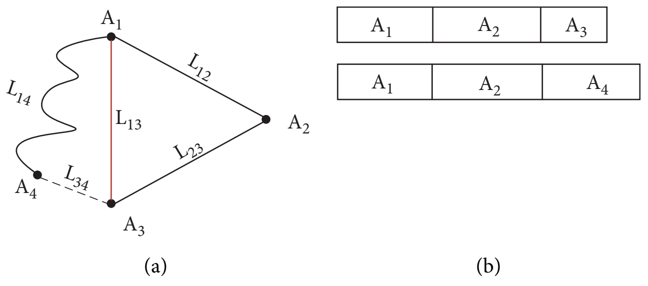


Figure 9: Scheduling scheme.

Based on the above-given assumptions and problem of existing SA, the selection of the initial executor set can be equivalent to the optimization problem:

$$\min H_E = \sqrt[m]{\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} H_{E_iE_j}^m},$$
$$s.t\forall E_i, E_j \in E, H_{E_iE_j}\Omega_{th}, \tag{21}$$
$$\Omega_{th} = \frac{\left(\max H_{E_iE_j} + \min H_{E_iE_j}\right)}{2}.$$

From (20), the heterogeneity between any two executors should be smaller than Ω_{th}. The ideal scheduling scheme is that the scheduling period is infinite, namely, the number of executors in the executor pool is infinite. But due to cost and environmental constraints in practical applications, the number of executors in the executor pool is often limited to less than 20. Therefore, the global optimal solution can be obtained in a short amount of time and the computational complexity is low.

Then, we consider the replacement of the executor. According to the feedback information of the ruling module, when the Local Confidence or Global Confidence is lower than the threshold or Adaptive Survival Time Disturbance Factor over time to live, it is necessary to schedule the executor offline and select a new executor online in time to further ensure the dynamic security of the system. In order to ensure the system service quality, we assume that only a single executor can be scheduled online or offline in one scheduling cycle. If two or more (but no more than half of the online executers) executors reach the offline threshold at the same time, the executor with the lowest HCE will be offline first. And, the operation will be repeated in the next scheduling cycle or cycles. If more than half of executors' HCEs reach the scheduling threshold, we assume the system suffers from serious loss due to the attack, and all online executors should be immediately taken offline for cleaning and recovery.

As shown in Figure 9(b), we consider a 3-redundancy mimic system, where the executor pool is $E = \{A_1, A_2, A_3, A_4\}$, and the length of $|A_i|$ represents the time period since executor E_i was scheduled offline the last time, also known as its scheduling period. The process of selecting a new online executor is shown in Figure 9(b). Assuming that the overall heterogeneity of $\{A_1, A_2, A_3\}$ is greater than that of $\{A_1, A_2, A_4\}$, an original system would be more inclined to choose A_3 to go online and ignore the potential threat of its overly short offline time. But system should be more inclined to choose executor A_4, because $|A_1| + |A_2| + |A_4||A_1| + |A_2| + |A_3|$. If the scheduling period is too short, it can cause an executor to go online too quickly, and give the attack more priori knowledge. We should try to avoid this situation by introducing a sleep threshold.

Because $|A_1| + |A_2| + |A_4||A_1| + |A_2| + |A_3|$. If the scheduling period is too short, it can cause an executor to go online too quickly and give the attack more priori knowledge. We should try to avoid this situation by introducing a sleep threshold.

Definition 10. (Sleep Threshold θ) Assuming that the last offline time of executor E_j is t_i, then $\Delta t = t_j - t_i$ represents the offline sleep time of executor E_j, and the sleep threshold is the optimal time, the system has the highest security and the lest cost. Supposing that the number of executors in the executor pool is n, and the redundancy of online executor set is m, we define the sleep threshold θ in this paper as follows ([] represents choosing integer):

$$\theta = \left[\frac{n-m}{2}\right]. \tag{22}$$

Definition 11. (Rotational Scheduling) Assuming online executor set is $E_{\text{on}} = \{E_1, E_2, \ldots E_m\}$, and the executor pool is $E_{\text{pool}} = \{E_l, E_{l+1}, \ldots E_{l+n}\}$. A rotational scheduling process is to roll an executer offline and roll another online. When the historical confidence of $E_i \in E_{\text{on}}$ is lower than the threshold, it will be rolled offline into the cleaning process, and a new executor $E_j \in E_{\text{pool}}$ will be selected to go online. After a rotational scheduling process, the new executor set will be $E_{\text{on}}' = (E_{\text{on}}/E_i) \cup E_j$. Obviously, the selection of a replacement executor is an optimization problem, and we formally formulate it as follows:

$$\min H_{E_j} = \alpha \sqrt[m]{\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} H_{E_iE_j}^m} + \beta H_{E_iE_j} \tag{23}$$
$$\left(E_a, E_j \in E_{\text{on}}', E_i \in E_{\text{on}}\right),$$

$$\forall E_a \in \frac{E_{\text{on}}}{E_i}, H_{E_jE_a}\Omega_{th}, \tag{24}$$

$$\forall E_j \in E_{\text{pool}}, t_i - t_{i-1}\theta, \tag{25}$$

$$\alpha + \beta = 1. \tag{26}$$

Equation (26) represents the similarity between $\forall E_a \in E_{\text{on}}/E_i$ and E_j is smaller than the threshold. Eq. (27) shows the scheduling period of E_j should be larger than the sleep threshold. (26) shows the weight of the objective function. From the above-given derivation, the pseudo-code of HHAC proposed is shown in Algorithm 2. Lines 1–9 select the initial online executor set according to the heterogeneity calculated by the HTVG. In lines 10–12, we compute the LC and Adaptive Survival Time Disturbance Factor of each executor. In lines 13–15, if the LC of E_i is lower than threshold CON_{th} or $\phi^{E_i}_{\text{daptive}}$ exceeds the sleeping threshold, E_i will be offline. From lines 16 to 20, according to the rotational scheduling rules and executor pool, the HHAC selects the replaced online executor. To the best of our knowledge, we are the first to consider the sleep threshold to select a new online executor. Compared with existing scheduling algorithms, the HHAC can achieve better system reliability in more complex attack scenarios.

6. Evaluation Model Based on AHP-FCE

Existing researches mainly rely on a single index, such as scheduling period, antiattack capability, system cost, and service quality to evaluate the pros and cons of a SA, which has a large deviation. In practical applications, we cannot just consider security without considering the possibility or cost of deployment, or only consider deployment cost without considering system security. Based on the above-given considerations, this paper introduces the AHP-FCE model for comprehensive SA evaluation based on multiple factors such as dynamics, heterogeneity metrics, system cost, and service quality.

6.1. Determine Indicator Weights Using AHP.

6.1. Determine Indicator Weights Using AHP. Analytic Hierarchy Process (AHP) [32] is a systematic and hierarchical analysis method that combines qualitative and quantitative methods. As shown in Table 3, this paper adopts Santy's 1–9 scale method to determine the degree of importance.

Theorem 1. *Eigenvalue method: if the pairwise comparison matrix is not a consistent matrix, the normalized eigenvector corresponding to its largest eigen root is used as the weight vector $W = \{w_1, w_2, \ldots w_n\}$, then $AW = \lambda W$. CI is the divergence between the judgment matrix and consistency, namely:*

$$CI = \frac{\lambda_{\max} - n}{n - 1}. \tag{27}$$

Considering the random consistency index RI, the consistency ratio CR can be calculated as follows:

$$CR = \frac{CI}{RI}. \tag{28}$$

When $CR0.1$, we consider A has a satisfying consistency, and the consistency test is passed. Otherwise, it is necessary to reconstruct the judgment matrix and retest.

6.2. FCE Model. Fuzzy Comprehensive Evaluation (FCE) [33, 34] is a comprehensive evaluation method based on fuzzy mathematics, which is suitable for solving fuzzy or difficult-to-quantify problems. Therefore, we apply FCE to the evaluation of SA. Here the trapezoidal distribution membership function used in this section is as follows:

$$U_iV_1 = \begin{cases} 1, & x \le 20, \\ \dfrac{40 - x,}{40 - 20,} & 20 < x < 40, \\ 0, & x \ge 40, \end{cases}$$

$$U_iV_j = \begin{cases} 0, & x \le a_{j-1}, \\ \dfrac{x - a_{j-1}}{a_j - a_{j-1}}, & a_{j-1} < x < a_j, \\ 1, & a_j < x < a_{j+1}, \\ \dfrac{a_{j+1} - x}{a_{j+1} - a_j}, & a_{j+1} < x < a_{j+2}, \\ 0, & x \ge a_{j+2}, \end{cases} \tag{29}$$

$$U_iV_5 = \begin{cases} 0, & x \le 60, \\ \dfrac{x - 60}{80 - 60}, & 60 < x < 80, \\ 1, & 80 < x < 100. \end{cases}$$

Then, weight $W = [w_1, w_2 \ldots w_n]$ of U can be obtained from the above AHP, then the evaluation matrix A can be calculated as follows:

$$\begin{aligned} A &= W \times R \\ &= [w_1, w_2 \ldots w_n] \begin{bmatrix} U_1V_1 & U_1V_2 & \cdots & U_1V_m \\ U_2V_1 & U_2V_2 & \cdots & U_2V_m \\ \vdots & \vdots & \ddots & \vdots \\ U_nV_1 & U_nV_2 & \cdots & U_nV_m \end{bmatrix}, \\ &= [a_1, a_2, \ldots, a_m], \end{aligned} \tag{30}$$

where R is the fuzzy matrix of evaluation factor set U about evaluation set V, and U_iV_j represents the membership score about V_j. At the same time, score set for each evaluation is $S = [s_1, s_2, \ldots, s_m]$. Finally, the comprehensive score $S_{\text{score}} = B \times S^T$ can be used to evaluate the pros and cons of the SA comprehensively.

```
     **Input:** Executor pool $E_{\text{pool}}$, Vulnerability Topology Graph VTG, Random Survival Time $\phi_{\text{random}}$, sleep threshold $\theta$
     **Output:** Online executor pool $E_{\text{on}}$, offline executor $E_i$, online executor $E_j$
     **Function** Select-First $(H_{E_iE_j})$
     / * **Select initial online executor set** * /
 (1) $\forall i, j \in (1, m), E_i, E_j \in E_{\text{pool}}$;
 (2) Compute heterogeneity threshold $\Omega_{th}$: $\Omega_{th} = (\max H_{E_iE_j} + \min H_{E_iE_j})/2$
 (3) If $i \neq j$andH$_{E_iE_j}\Omega_{th}$ then
 (4) $H_{E_{\text{on}}} \leftarrow \sqrt[m]{\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} H_{E_iE_j}^{m}}$
 (5) For each $H_{E_{\text{on}}}{}'$ in $H_E$
 (6) If $H_{E_{\text{on}}}{}' < \min H_{E_{\text{on}}}$ then
 (7) $\min H_{E_{\text{on}}} \leftarrow H_{E_{\text{on}}}{}'$
 (8) Endif
 (9) Return $\min H_{E_{\text{on}}}$
     / * **Select offline executor** * /
     **Function** offline executor $(\phi_{\text{random}}, \text{CON}_{\text{local}})$
(10) For each $E_i$ in $E_{\text{on}}$:
(11) $\text{CON}_{\text{local}}^{E_i} \leftarrow \text{CON}_{\text{global}}^{E_i} + \text{d}(\text{CON}_{\text{local}})/\text{d}t$
(12) $\phi_{\text{daptive}}^{E_i} \underset{E}{\leftarrow} \phi_{\text{random}}^{E_i} + f(t)$
(13) If $\text{CON}_{\text{local}}^{E_i}\text{CON}_{th}$ or $t_{E_i}\phi_{\text{daptive}}^{E_i}$:
(14) $E_i$ offline
(15) Return $E_i$
     / * **Select replaced online executor** * /
     **Function** Select-New $(H_{E_iE_j}, \Delta t, \Omega_{th}, \theta)$
(16) For $E_j$ in $E_{\text{pool}}/E_{\text{on}}$, $E_i$ in $E_{\text{on}}$:
(17) If $H_{E_iE_j}\Omega_{th}$ and $\Delta t\theta$ then
(18) $\min H_{E_j} = \alpha \sqrt[m]{\sum_{i=1}^{m-1} \sum_{j=i+1}^{m} H_{E_iE_j}^{m} + \beta H_{E_iE_j} (\alpha + \beta = 1)}$
(19) End if
(20) Return $E_j$
     / * **Compute heterogeneity of executors** * /
     **Function** HTVG (VTG)
(21) Return $H_{E_iE_j}$, $H_E$
```

Algorithm 2: Scheduling algorithm based on high-order heterogeneity and confidence (HHAC).

6.3. Evaluation Criteria. By researching and summarizing the existing scheduling algorithms and evaluation indexes, this paper formulates the index evaluation criteria table on the basis of the calculation methods of related evaluation indexes, as shown in Table 4.

7. Experiment Result

In this section, we demonstrate the effectiveness of the scheduling algorithm HHAC proposed in this paper through simulation experiments. Firstly, we introduce the setup of experimental simulation and then analyze the simulation results.

7.1. Experimental Environment. Specifically, the configuration of the server includes AMD 4800U, 1.8 GHz, and 16 GB RAM. The operation platform is Pycharm2020. In this experiment, we set the redundancy of online executions to be 3 and 4, respectively, and the number of executors in the executor pool is 15. In order to reduce the uncertainty, the experiment is repeated 120 times to obtain the final evaluations of four scheduling algorithms. In addition, to better focus on the actual performance of the scheduling algorithm, we simplify some experimental conditions and parameter settings:

(i) During the experiment, in all four algorithms, we generate the heterogeneity between executors $(H_{E_iE_j})$ using the same normal distribution with parameters $(0, 0.5)$ and the same random seed. And, the failure probability of executor λ is generated by a normal distribution with parameters $(0, 0.1)$.

(ii) This experiment mainly researches the selection of online executors, and we simplify the selection of offline executors into selecting offline executors based on the probability of failure.

(iii) Here, we assume that only one executor can be rolled online or offline in one scheduling cycle to improve the service quality.

(iv) In order to better research the dynamics of the scheduling algorithm and the average failure probability of the system, we do not change the online executor redundancy in a single scheduling experiment.

7.2. Comparison Indexes. Existing research to evaluate scheduling algorithms mainly include the system dynamics [19, 31] and the probability of system failure [17, 35]. In addition to the above-given two indicators, this paper innovatively proposes the minimum sleep

Table 3: Definitions of Saaty's 9-point scale grades.

a_{ij}	Definition
1	Factors i and j have the same degree of importance
3	The importance of factor i is slightly higher than factor j
5	The importance of factor i is significantly higher than factor j
7	The importance of factor i is much higher than factor j
9	The importance of factor i is extremely higher than factor j
2, 4, 6, 8	Median value between the two adjacent values

Table 4: Index evaluation criteria.

Index	Evaluation criteria
Dynamic	Scheduling times between the same scheduling scheme
Heterogeneity	(1) Known common vulnerability (30%) (2) Unknown vulnerability (20%) (3) High-order vulnerability (20%) (4) Method of executor set heterogeneity (20%)
Cost	(1) Time complexity of scheduling algorithm (30%) (2) Space complexity of scheduling algorithm (30%) (3) Scheduling frequency of executor (40%)
Quality of service	(1) Recovery time of single scheduling (30%) (2) Quantity of single scheduling (30%) (3) Scheduling times between same time period (40%)

threshold of executor, the cumulative distribution function of system failure, and minimum sleep times to further illustrate the effectiveness of the HHAC proposed in this paper.

(i) Dynamics of scheduling algorithm S: the number of scheduling times between the same scheduling scheme is generated, regardless of the position of executor.

(ii) Minimum sleep threshold of executor θ: the minimum number of scheduling times of the executor set during the time when the executor is rolled offline until it is scheduled to go online again.

(iii) System average failure probability $\overline{\tau}$: the system failure probability proposed in [35] is related to the failure probability of single executor and the similarity between executors. Based on [35], this paper proposes the system average failure probability further based on the number of failure executors and the average minimum sleep threshold of the single executor as follows:

$$\tau_c = \sum_{i_0=1}^{n} \sum_{i_1=i+1}^{n} \cdots \sum_{i_j=i+j}^{n} \lambda_{i_0} \cdot \lambda_{i_1} \cdot \cdots \cdot \lambda_{i_j} \cdot \frac{\left(\sum_{k=i_0}^{i_j} \sum_{l=i_0+1}^{i_j} H_{E_k E_l}\right)}{\sum_{i=1}^{n} \sum_{j=i+1}^{n} H_{E_i E_j}} \left(i_j = \left[\frac{n}{2} + 1\right]\right),$$
$$\overline{\tau} = \frac{\sum_{c=1}^{S} \tau_c}{(S \cdot \overline{\theta})}. \qquad (31)$$

7.3. Compared Schemes. In this experiment, we compare the HHAC comprehensively with the typical Completely random scheduling algorithm (CRS), Time-based random threshold scheduling algorithm (TIRTS), and the Random seed and minimum similarity (RSMS) to help readers of our paper better understand our algorithm and its advantage.

7.3.1. CRS. This scheme randomly selects the online and offline executors.

7.3.2. TIRTS. This scheme assigns a random online time to the executor, and when the online time of the executor online reaches this time, rolls the executor offline for cleaning, and then randomly selects an executor to go online.

7.3.3. RSMS [17]. This scheme first randomly selects the seed executor and then selects the scheduling scheme with the smallest overall similarity according to the principle of minimum similarity.

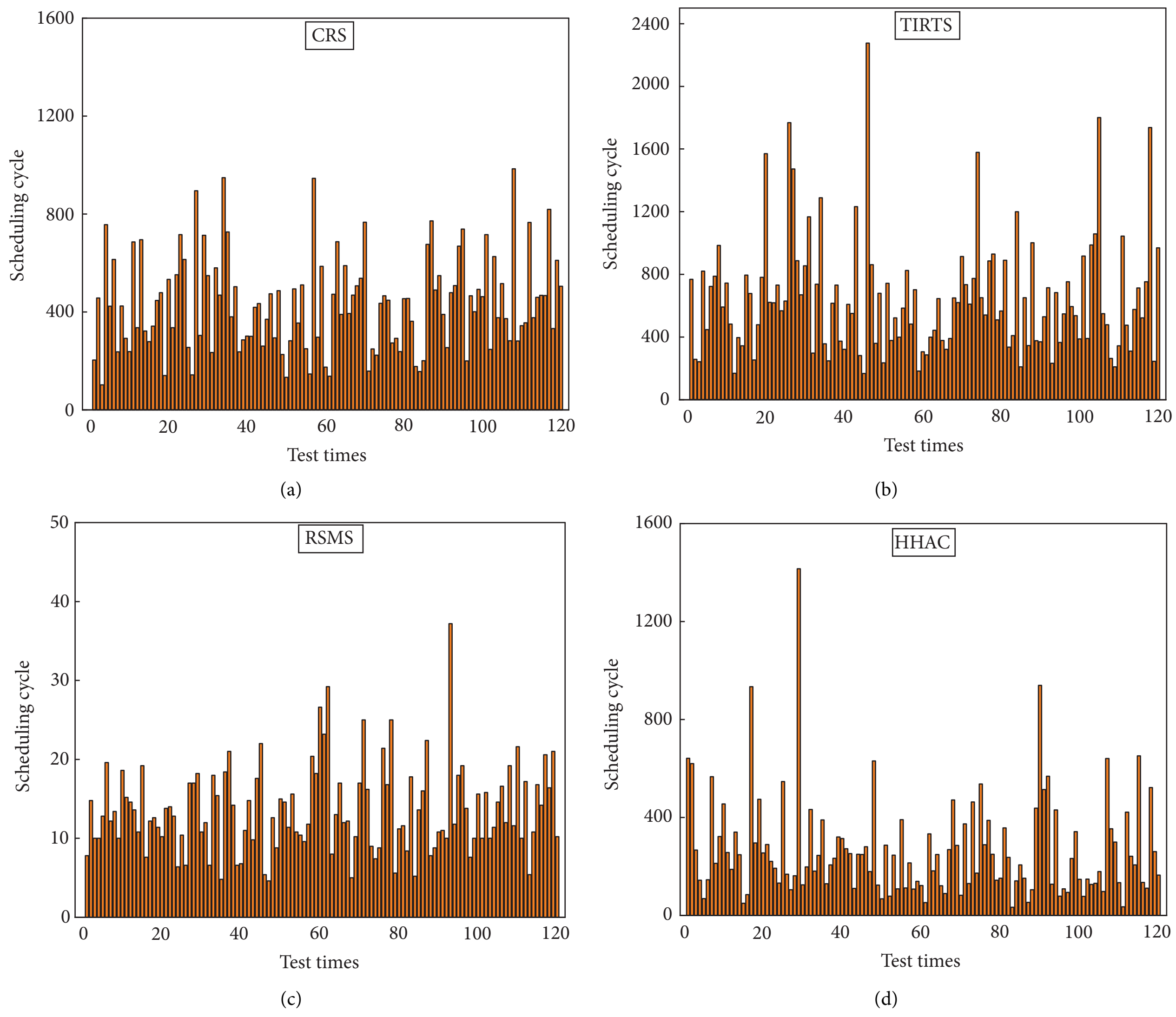


FIGURE 10: Scheduling cycle with redundancy $n = 3$. (a) CRS, (b) TIRTS, (c) RSMS, (d) HHAC.

7.4. Simulation Results

7.4.1. System Dynamics. We first compare the dynamics of our algorithm with CRS, TIRTS, and RSMS. In order to avoid accidental errors, we program all algorithms with *Python* scripts and run every experiment 120 times. In order to compare the scheduling cycles under different redundancy, this paper selects different redundancy with 3 and 4. Figure 10 shows the scheduling cycles of four scheduling algorithms with online executor redundancy $n = 3$, and Figure 11 shows the results with the online executor redundancy $n = 4$. Under different redundancy, the average scheduling period of the four algorithms is shown in Table 5.

It can be seen that, along with the increasing of online executor redundancy, the scheduling cycle of HHAC gradually increases but still less than the CRS and TIRTS algorithms. For example, the scheduling cycle of CRS and TIRTS is about 2 times than HHAC, and HHAC is about 20 times than RSMS when $n = 3$. When $n = 4$ the scheduling cycle of HHAC is about 28 times than RSMS, and 25% to CRS and TIRTS. This result is easy to understand. In CRS and TIRTS, each executer is selected randomly, while HHAC adds many restrictions to reduce the scheduling cycle. In addition, we can see the scheduling cycle of CRS and TIRTS are relatively scattered and random, and HHAC has a relatively centralized scheduling cycle. Hence, in a limited system considering security and stability, HHAC could be a good choice to achieve high security and reliability.

7.4.2. Minimum Sleep times of Executor. In this simulation to further verify the feasibility of HHAC, we evaluate minimum sleep times by calculating the average minimum sleep times of each executor. We run every experiment 120 times. The same Figure 12(a) shows the average minimum sleep times of each algorithm when $n = 3$, and Figure 12(b) shows the average minimum sleep times of the four algorithms when $n = 4$. It is observed from above figure that the minimum sleep times of HHAC is significantly larger than that of CRS and TIRTS. More specific, the average

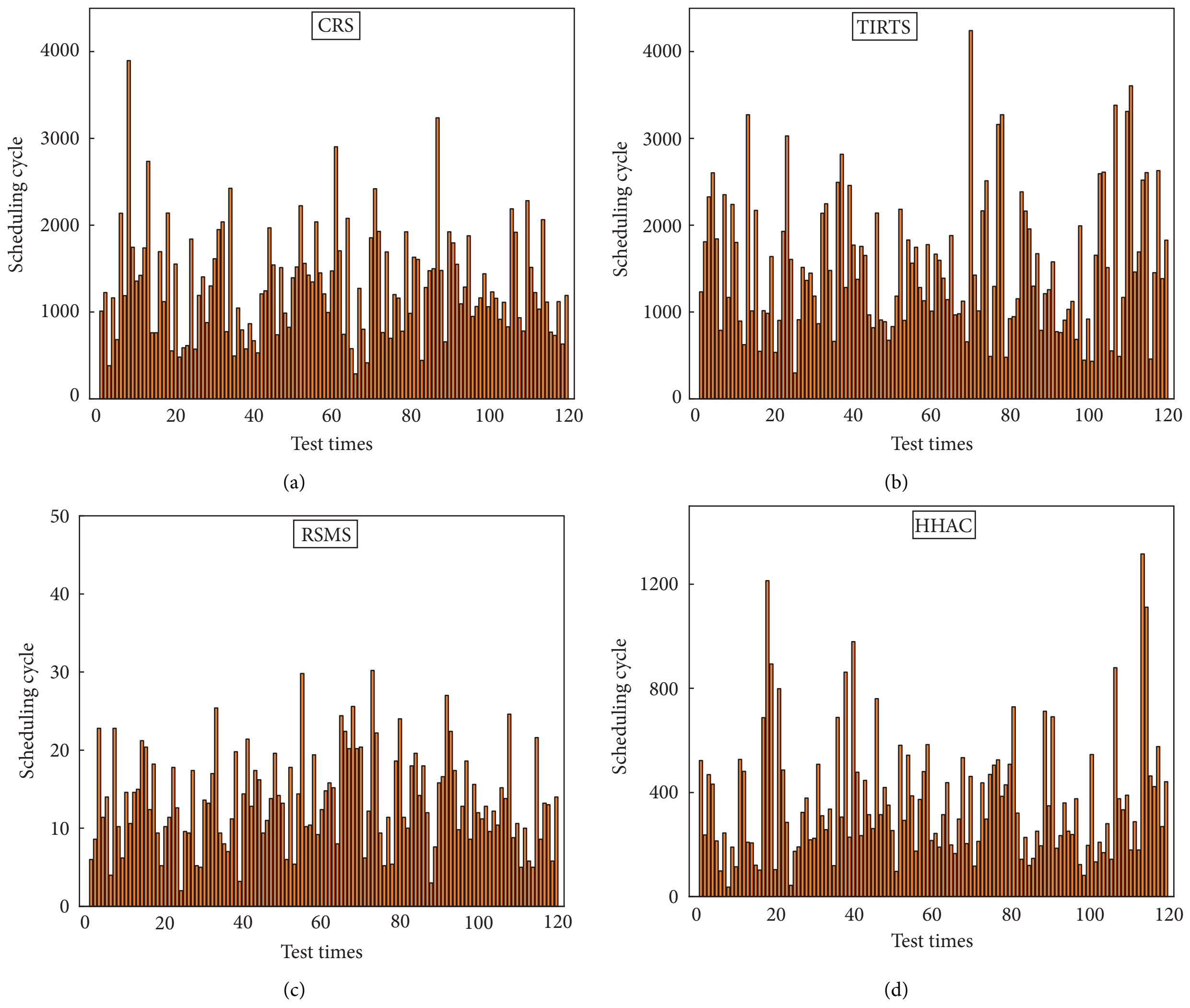


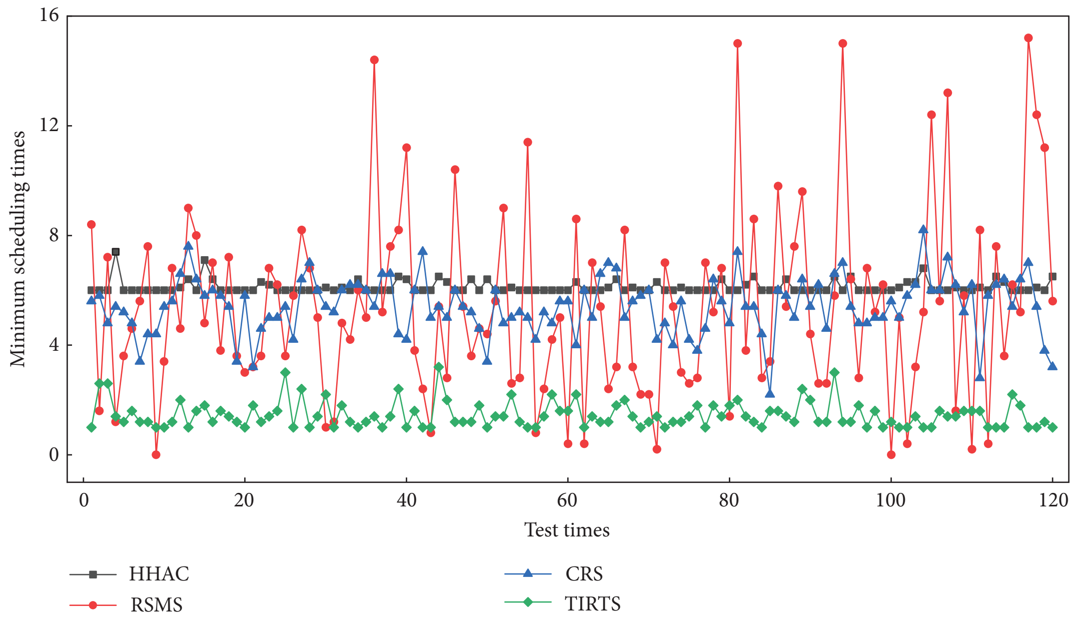
FIGURE 11: Scheduling cycle with redundancy $n = 4$. (a) CRS, (b) TIRTS, (c) RSMS, (d) HHAC.

TABLE 5: Scheduling cycle.

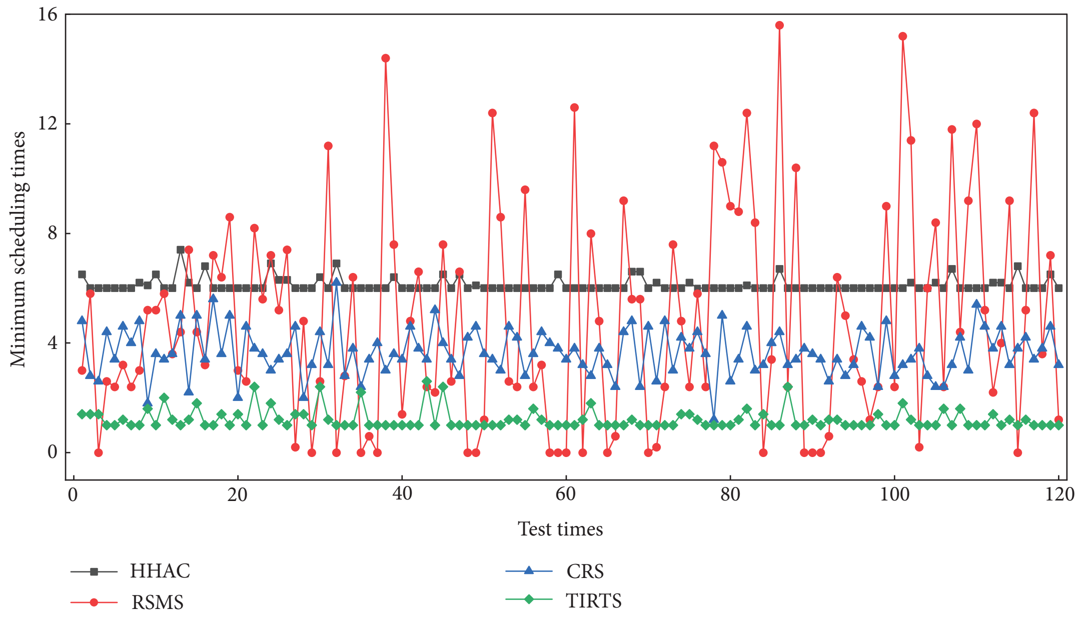
Redundancy	Algorithm			
	CRS	TIRTS	RSMS	HHAC
$n = 3$	435	649	13	267
$n = 4$	1317	1538	13	366

minimum sleep times of HHAC is nearly 12.25% more than RSMS and CRS, and 5 times than TIRTS when $n = 3$. In addition, the average minimum sleep times of HHAC is nearly 26.39% more than RSMS and CRS when $n = 4$. Although the average minimum sleep times of HHAC is lower than RSMS in some times, the RSMS is relatively random, which will greatly affect the security and reliability of the system. In summary, in terms of the minimum sleep times of executor, HHAC achieves better performance compared to the other three algorithms. With the greater sleep times, HHAC can better avoid the attacker's long-term system scanning increase the system security.

7.4.3. System Failure Probability. We evaluate the average failure probability of the system by calculating the average failure probability of each online executor set between the same scheduling scheme. Figure 13(a) shows the system failure probabilities of four algorithms when $n = 3$, and the system failure probabilities under redundancy $n = 4$ in Figure 13(b). Simultaneously, we summarize the average failure probability of four algorithms in Table 6. Obviously, compared to the other three algorithms, HHAC can achieve better results on system security under the same redundancy. Specifically, the average failure probability of HHAC is 28.8% less than RSMS, 4.5% less than CRS when $n = 3$, and much less than TIRTS. In addition, the average failure probability of HHAC is 56% less than RSMS under redundancy $n = 4$. Intuitively, with the increase of executor redundancy, the average failure probability of the system gradually decreases around 10 times, which is in line with the security changing rule of the mimic system. In summary, compared to the other three algorithms, HHAC maybe a better choice to guarantee system security under certain constraints.



(a)

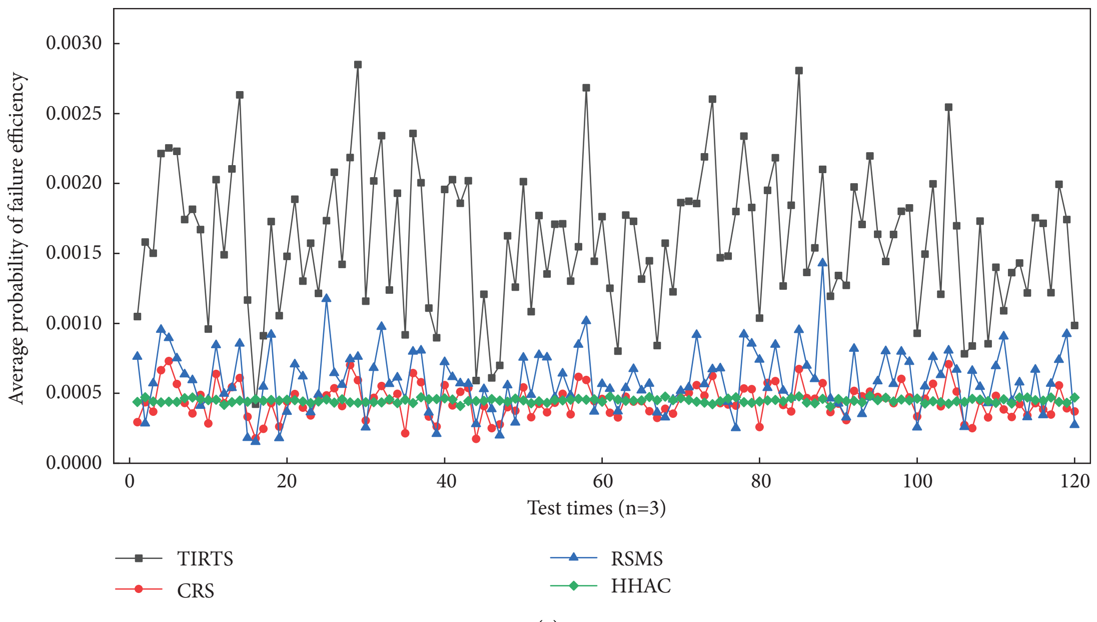


(b)

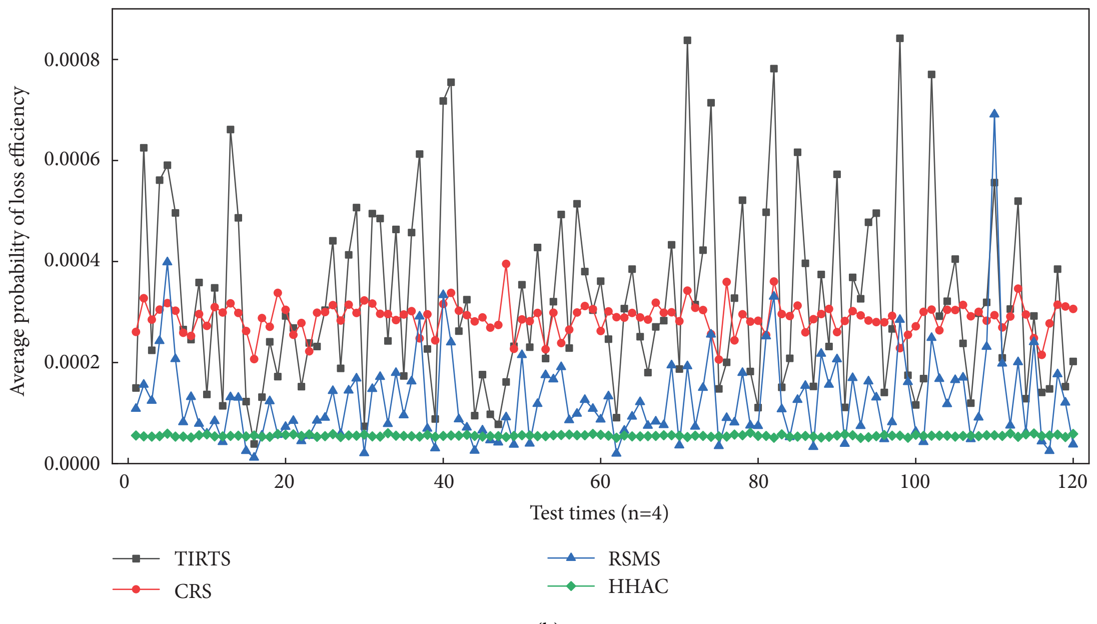
Figure 12: Minimum sleep times with different redundancy. (a) $n = 3$, (b) $n = 4$.

7.4.4. The CDF of Minimum Sleep times of Executor. It can be seen from Figure 14 that when the redundancy of online executor is $n = 3$, the minimum sleep times of HHAC is stable at about 6 times, sometimes less than RSMS and CRS. But the above-given two algorithms are very unstable and less than HHAC in most of time. Moreover, compared with TIRTS, HHAC has great advantages to ensure system security. Similarly, HHAC has the same advantage compared to TIRTS and CRS when redundancy $n = 4$, but less than RSMS to a certain degree.

7.4.5. The CDF of System Failure Probability. Figure 15(a) shows the CDF of system failure probability when $n = 3$, it is observed from the above-given figure that the system failure probability of HHAC is much smaller than TIRTS and less than RSMS and CRS in about 50% of experiments. Moreover, compared with RSMS and CRS, the system failure probability of HHAC is very stable and most around 0.42%. The same compared to TIRTS and CRS when $n = 4$, HHAC has an obvious advantage, but larger than RSMS on some times.



(a)

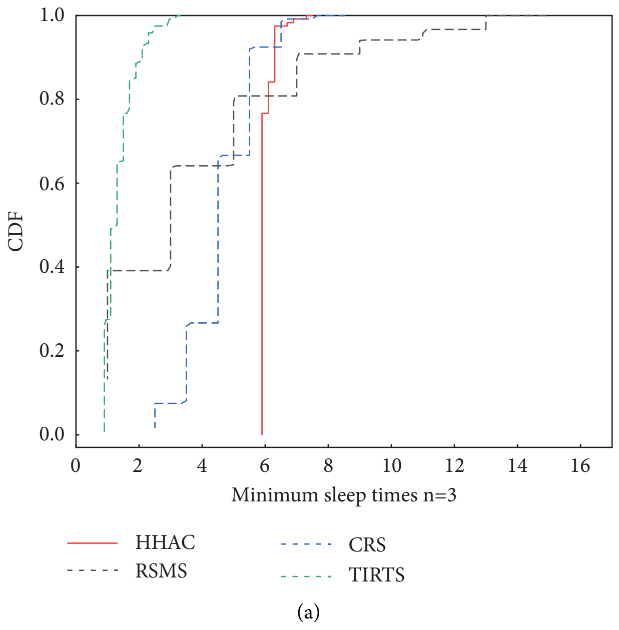


(b)

FIGURE 13: Failure probability with different redundancy. (a) $n = 3$, (b) $n = 4$.

TABLE 6: Failure probability.

Redundancy	Algorithm			
	CRS (%)	TIRTS (%)	RSMS (%)	HHAC (%)
$n = 3$	0.44	1.598	0.592	0.42
$n = 4$	0.289	0.323	0.125	0.055



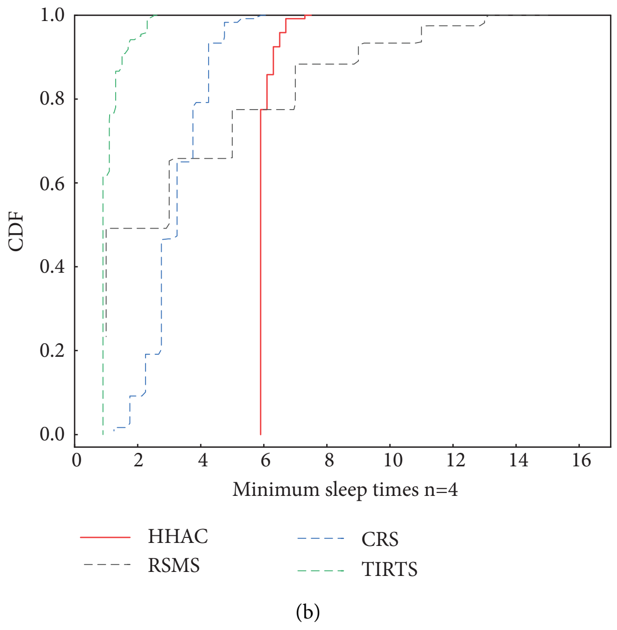
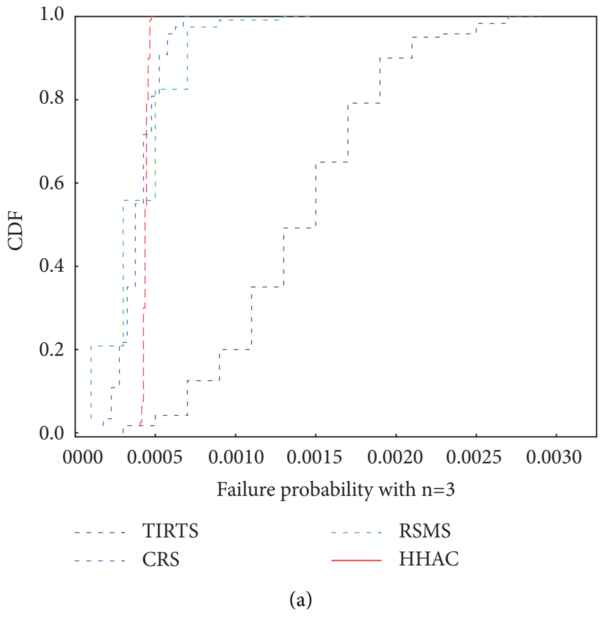


Figure 14: The CDF of minimum sleep times under different redundancy. (a) $n = 3$, (b) $n = 4$.



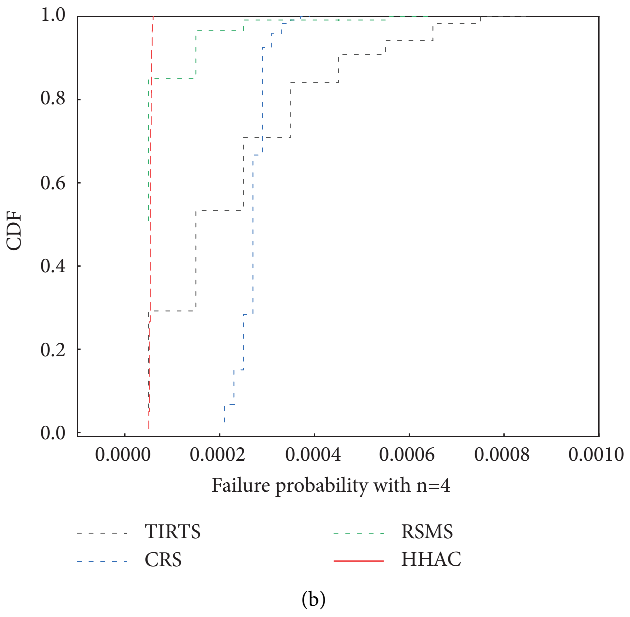


Figure 15: The CDF of system failure probability under different redundancy. (a) $n = 3$, (b) $n = 4$.

7.4.6. Comprehensive Evaluation Based on AHP-FCE

(1) Constructing judgment matrix. In order to ensure the objectivity of evaluation on scheduling algorithms, we determine the relationship between dynamics, heterogeneity, system cost, and quality of service by adopting Santy's 1–9 scaling method. Firstly, since the main goal of the scheduling algorithm is to construct a system with an unmeasurable internal structure and high security, the system security should be the primary goal when constructing the judgment matrix while balancing the deployment cost. Considering the above-given reasons, the judgment matrix A is determined in this paper as follows:

$$A = \begin{pmatrix} 1 & 1 & 3 & 2 \\ 1 & 1 & 3 & 2 \\ \frac{1}{3} & \frac{1}{3} & 1 & 1 \\ \frac{1}{2} & \frac{1}{2} & 1 & 1 \end{pmatrix}. \tag{32}$$

(2) Calculating the weight vector. According to the above-given matrix, the weight vector ω of dynamics, heterogeneity, system cost and QoS calculated by AHP is

$$\omega = (0.354, 0.354, 0.131, 0.161). \tag{33}$$

(3) Comprehensive evaluation. According to the research on CRS, TIRTS, RSMS, and HHAC and the comprehensive evaluation table in Table 4, we obtain the comprehensive scores of four algorithms and show them in Table 7. According to the membership function, the fuzzy matrices of CRS, TIRTS, RSMS, and HHAC can be obtained as follows:

$$\begin{aligned} R_1 &= \begin{pmatrix} 0 & 0 & 0 & 0.35 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0.2 & 1 & 0.8 & 0 & 0 \\ 0 & 0 & 0.9 & 1 & 0.1 \end{pmatrix}, \\ R_2 &= \begin{pmatrix} 0 & 0 & 0 & 0.3 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0.85 & 1 & 0.15 & 0 \\ 0 & 0 & 0.75 & 1 & 0.25 \end{pmatrix}, \\ R_3 &= \begin{pmatrix} 0 & 0.4 & 1 & 0.6 & 0 \\ 0 & 0.25 & 1 & 0.75 & 0 \\ 0 & 0 & 0.25 & 1 & 0.75 \\ 0 & 0.65 & 1 & 0.35 & 0 \end{pmatrix}, \\ R_4 &= \begin{pmatrix} 0 & 0 & 0.6 & 1 & 0.4 \\ 0 & 0 & 0.5 & 1 & 0.5 \\ 0 & 0 & 0.9 & 1 & 0.1 \\ 0 & 0 & 0.4 & 1 & 0.6 \end{pmatrix}. \end{aligned} \tag{34}$$

Combined with ω we can know

Table 7: Comprehensive scoring table.

Algorithm	Score			
	Dynamic	Heterogeneity	Cost	QoS
CRS	87	0	36	62
TIRTS	86	0	43	65
RSMS	52	65	55	47
HHAC	68	70	62	72

$$\begin{aligned} B &= \begin{pmatrix} \omega \cdot R_1 \\ \omega \cdot R_2 \\ \omega \cdot R_3 \\ \omega \cdot R_4 \end{pmatrix} \\ &= \begin{pmatrix} 0.38 & 0.131 & 0.25 & 0.285 & 0.37 \\ 0.354 & 0.111 & 0.252 & 0.287 & 0.394 \\ 0 & 0.335 & 0.902 & 0.665 & 0.098 \\ 0 & 0 & 0.572 & 1 & 0.428 \end{pmatrix}. \end{aligned} \tag{35}$$

In this paper, we define the comment set as S = {very poor, poor, average, good, excellect}, = {10, 20, 30, 40, 50}, and the final score is calculated as follows:

$$\begin{aligned} S_{\text{score}} &= B \cdot S^T \\ &= (43.82, 44.5, 65.26, 78.56). \end{aligned} \tag{36}$$

From the final score of the AHP-FCE evaluation model, the RSMS and HHAC have achieved better results in system security compared with CRS and TIRTS. This result is easy to understand. For CRS and TIRTS lack consideration of executor heterogeneity, which may lead to many potential security threats in the online executors. In addition, considering the system cost and service quality, the HHAC proposed in this paper has a higher score than RSMS and achieves better results. To sum up, the AHP-FCE comprehensive evaluation model proposed in this paper can evaluate the merits of each algorithm in a more comprehensive and fine-grained way. Moreover, the AHP-FCE evaluation model fills the gap in the comprehensive evaluation of scheduling algorithms and achieves better results. Constrained by our limited understanding of scheduling algorithms, we only formulate the scoring rule table for four algorithms. In the future, we intend to build more complete scoring rules and perfect the AHP-FCE model.

8. Conclusions

Mimic Defense has gradually shown powerful defense capabilities in cyberspace security. In this paper, we focus on the research and implementation of SA and has achieved some results. Firstly, considering the high-order common vulnerabilities among executors, this paper proposes a new executor heterogeneity quantification algorithm HVTG based on graph theory, which realizes the fine-grained heterogeneity measurement; secondly, we propose an adaptive scheduling algorithm HHAC based on high-order heterogeneity and historical confidence of executor and makes some result in security and reliability of the mimic system. For example, the minimum sleep times of HHAC is nearly 12.25% more than RSMS and CRS when $n = 3$, and the failure probability of HHAC is about 28.8% lower than RSMS under redundancy $n = 3$. Finally, we propose a comprehensive evaluation index, and evaluation model of the scheduling algorithm based on the AHP-FCE. The evaluation model fills the gap of a comprehensive evaluation of scheduling algorithm, and the result shows the HHAC has a good result than other algorithms through the evaluation model.

Looking into the future, we plan to conduct more in-depth and comprehensive research on the adaptive historical confidence, adaptive online redundancy, and evaluation criteria in the AHP-FCE evaluation model; for example, we will research how to continuously change the number of online executors, and eventually achieve high security balancing the system cost and quality of service.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Nuclear Foundation Major Special Fund (Grant no. 2017ZX01030301).

References

[1] S. Wei and H. Zhang, "Majority voting algorithm and performance analysis based on high level heterogeneity," *Computer Engineering*, vol. 51, no. 1, pp. 1–7, 2020.

[2] J. X. Wu, "Research on cyber mimic defense," *Journal of Cyber Security*, vol. 1, no. 4, pp. 1–10, 2016.

[3] H. Hu, Z. Wang, G. Cheng, and J. Wu, "MNOS: a mimic network operating system for software defined networks," *IET Information Security*, vol. 11, no. 6, pp. 345–355, 2017.

[4] G. Li, W. Wang, K. Gai, Y. Tang, B. Yang, and X. Si, "A framework for mimic defense system in cyberspace," *Journal of Signal Processing Systems*, vol. 93, no. 2-3, pp. 169–185, 2021.

[5] W. Wang, G. Li, and K. Gai, "Modelization and analysis of dynamic heterogeneous redundant system," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 12, pp. 35–51, 2022.

[6] W. C. Li, Z. Zhang, and J. X. Wu, "The modeling and risk assessment on redundancy adjudication of mimic defense," *Journal of Cyber Security*, vol. 3, no. 5, pp. 64–74, 2018.

[7] H. L. Ma, Y. M. Jiang, and B. Bai, "Tests and analyses for mimic defense ability of routers," *Journal of Cyber Security*, vol. 2, no. 1, pp. 43–53, 2017.

[8] K. Song, Q. R. Liu, and S. Wei, "Endogenous security architecture of Ethernet switch based on mimic defense," *Journal on Communications*, vol. 41, no. 5, pp. 18–26, 2020.

[9] Z. B. Zhu, Q. R. Liu, and D. P. Liu, "Research progress of mimic multi-execution scheduling algorithm," *Journal of Communications*, vol. 42, no. 5, pp. 179–190, 2021.

[10] Z. Y. Gu, X. M. Zhang, and S. J. Lin, "Load-aware dynamic scheduling mechanism based on security strategies," *Journal of Computer Applications*, vol. 37, no. 11, pp. 3304–3310, 2017.

[11] H. Y. Jia, Y. F. Pan, and W. H. Liu, "Executive dynamic scheduling algorithm based on high-order heterogeneity," *Journal on Communications*, vol. 43, no. 3, pp. 233–245, 2022.

[12] X. M. Wang, W. H. Yang, and W. Zhang, "Research on scheduling strategy of mimic Web server based on BSG," *Journal on Communications*, vol. 39, no. S2, pp. 112–120, 2018.

[13] Q. Lin, Y. Zhang, S. Verwer, and J. Wang, "MOHA: a multi-mode hybrid automaton model for learning car-following behaviors," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 2, pp. 790–796, 2019.

[14] C. Xing and C. Yuan, "Construction of optimal (r, δ)-locally recoverable codes and connection with graph theory," *IEEE Transactions on Information Theory*, vol. 68, no. 7, pp. 4320–4328, 2022.

[15] W. B. Yao and X. Z. Yang, "Design of selective algorithm for diverse software components," *Journal of Harbin Institute of Technology*, vol. 35, no. 3, pp. 261–264, 2003.

[16] Y. Y. Lu, Y. F. Guo, and Z. P. Wang, "Negative feedback scheduling algorithm based on historical information in SDN," *Chinese Journal of Network and Information Security*, vol. 4, no. 6, pp. 45–51, 2018.

[17] Q. R. Liu, S. J. Lin, and Z. Y. Gu, "Heterogeneous redundancies scheduling algorithm for mimic security defense," *Journal on Communications*, vol. 39, no. 7, pp. 188–198, 2018.

[18] J. X. Zhang, J. M. Pang, and Z. Zhang, "Executors scheduling algorithm for Web server with mimic structure," *Computer Engineering*, vol. 45, no. 8, pp. 14–21, 2019.

[19] L. M. Pu, S. X. Liu, and R. H. Ding, "Heterogeneous executor scheduling algorithm for mimic cloud service," *Journal on Communications*, vol. 41, no. 3, pp. 17–24, 2020.

[20] Z. Q. Wu and J. Wei, "Heterogeneous executors scheduling algorithm for mimic defense systems," in *Proceedings of the 2019 IEEE 2nd International Conference on Computer and Communication Engineering*, pp. 279–284, IEEE Press, Piscataway, 2019.

[21] Z. P. Lu, F. C. Chen, G. Z. Cheng, C. Qi, and J. Ai, "Towards a dynamic controller scheduling-timing problem in software-defined networking," *China Communications*, vol. 14, no. 10, pp. 26–38, 2017.

[22] W. Guo, Z. Q. Wu, F. Zhang, and J. Wu, "Scheduling sequence control method based on sliding window in cyberspace mimic defense," *IEEE Access*, vol. 8, no. 5, pp. 1517–1533, 2020.

[23] S. Wei, H. Yu, and Z. Y. Gu, "Architecture of mimic security processor for industry control system," *Journal of Cyber Security*, vol. 2, no. 1, pp. 54–73, 2017.

[24] C. Qi, J. X. Wu, H. C. Hu, and G. Cheng, "Dynamic-scheduling mechanism of controllers based on security policy in software-defined network," *Electronics Letters*, vol. 52, no. 23, pp. 1918–1920, 2016.

[25] J. F. Li, *Research on Key Technologies of Mimic Defense in Software-Defined Network*, Information Engineering University, Zhengzhou, 2019.

[26] M. Gao, J. Luo, and H. Y. Zhou, "A differential feedback scheduling decision algorithm based on mimic defense," *Telecommunications Science*, vol. 36, no. 5, pp. 73–82, 2020.

[27] B. P. Kavin, S. Ganapathy, and A. Karman, “An intelligent task scheduling approach for cloud using IPSO and A search algorithm,” in *Proceedings of the 2018 eleventh international conference on contemporary computing (IC3)*, pp. 1–5, IEEE, Noida, India, August 2018.

[28] S. Gunasekaran, L. SaiRamesh, and S. Sabena, “Dynamic scheduling algorithm for reducing start time in Hadoop[C],” in *Proceedings of the International Conference on Informatics and Analytics*, pp. 1–4, New York, NY, USA, August 2016.

[29] S. Muthurajkumar, M. Vijayalakshmi, A. Kannan, and S. Ganapathy, “Optimal and energy efficient scheduling techniques for resource management in public cloud networks,” *National Academy Science Letters*, vol. 41, no. 4, pp. 219–223, 2018.

[30] P. K. Manadhata and J. M. Wing, “An attack surface metric,” *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, 2011.

[31] W. Zhang, S. Wei, and L. Tian, “Scheduling algorithm based on heterogeneity and confidence for mimic defense,” *Journal of Web Engineering*, pp. 971–998, 2020.

[32] T. L. Saaty, “Decision-making with the AHP: w,” *European Journal of Operational Research*, vol. 145, no. 1, pp. 85–91, 2003.

[33] J. F. Chen, H. N. Hsieh, and Q. H. Do, “Evaluating teaching performance based on fuzzy AHP and comprehensive evaluation approach,” *Applied Soft Computing*, vol. 28, no. 2, pp. 100–108, 2015.

[34] X. Wei, X. Luo, Q. Li, J. Zhang, and Z. Xu, “Online comment-based hotel quality automatic assessment using improved fuzzy comprehensive evaluation and fuzzy cognitive map,” *IEEE Transactions on Fuzzy Systems*, vol. 23, no. 1, pp. 72–84, 2015.

[35] Q. Li, S. Meng, X. Sang et al., “Dynamic scheduling algorithm in cyber mimic defense architecture of volunteer computing,” *ACM Transactions on Internet Technology*, vol. 21, no. 3, pp. 1–33, 2021.