Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 5750646, 17 pages
https://doi.org/10.1155/2021/5750646

*Research Article*

# An Approach to Modeling and Analyzing Reliability for Microservice-Oriented Cloud Applications

**Zheng Liu**[1], **Guisheng Fan**[1], **Huiqun Yu**[1] and **Liqiong Chen**[2]

[1]*Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai, China*
[2]*School of Computer Science and Information Engineering, Shanghai Institute of Technology, Shanghai, Shanghai, China*

Correspondence should be addressed to Guisheng Fan; gsfan@ecust.edu.cn

Received 4 May 2021; Accepted 21 July 2021; Published 26 August 2021

Academic Editor: Xiaoxian Yang



Microservice architecture is a cloud-native architectural style, which has attracted extensive attention from the scientific research and industry communities to benefit independent development and deployment. However, due to the complexity of cloud-based platforms, the design of fault-tolerant strategies for microservice-oriented cloud applications becomes challenging. In order to improve the quality of service, it is essential to focus on the microservice with more criticality and maximize the reliability of the entire cloud application. This paper studies the modeling and analysis of service reliability in the cloud environment. Firstly, a formal description language is defined to model microservice, user request, and container accurately. Secondly, the reliability analysis is conducted to measure a critical microservice's fluctuation and vibration attributes within a period, and the related properties of the constructed model are analyzed. Thirdly, a fault-tolerant strategy with redundancy operation has been proposed to optimize cloud application reliability. Finally, the effectiveness of the method is verified by experiments. The simulation results show that the algorithm obtains the maximum benefits and has high performance through several experiments.

## 1. Introduction

Cloud computing is becoming the mainstream of information technology, which usually involves providing dynamic, scalable, and virtual resources through the Internet. Cloud applications are usually large in scale, have complex structures and contain many distributed components. With the extensive deployment of information systems in critical industries and the continuous expansion of cloud computing applications, the demand for high reliability and cloud computing availability is becoming more urgent. Fault-tolerant technology is an effective way to ensure reliability using the redundancy method to eliminate the influence of fault. A new Fault-Tolerant Elastic Scheduling Algorithm FESTAL for real-time tasks is designed to promote resource utilization [1]. Research on fault-tolerant cloud computing technology can improve the performance so that the task being processed will not terminate abnormally or complete the constraint time functions.

Many of the existing studies on cloud workflow systems focus on reducing the budget and ensuring effectiveness by identifying the critical services in the cloud environment. Determining the potential costs of cloud computing can be complex considering the cloud application reliability. The heuristic search determines the most potential mobile edge computing nodes for each IoT service to meet the reliability requirement based on priorities [2]. There have been attempts to build neural network models for Quality of Service (QoS) prediction. The deep neural networks have achieved exciting results in improving QoS in cloud applications by alleviating overfitting problems. It is necessary to identify the components in cloud applications with high failure risk and reduce the adverse effects [3]. However, the existing methods are limited due to the lack of modeling and analysis of fluctuation reliability time series. Hence, providing highly reliable cloud applications is a challenging and critical research issue.

Microservice architecture offers an alternative solution to realizing software modularization, which has the enormous advantages of solid substitutability, sustainable development, independent scalability, and sustainable delivery. Due to the complexity of applications in the cloud-based system,

services are often completed by multiple microservices in complex networks. Microservices bring benefits for implementation flexibility, smooth scaling, improved delivery speed, and flexibility. It overcomes the shortcomings of the high deployment cost with traditional monomer and inflexible response to environmental changes. For Quality of Experience (QoE) requirements, the microservice-oriented application needs long-term stable implementation due to environmental changes [4]. Hence, how to guarantee the reliability of microservice-based cloud applications is considered a challenging problem.

To solve these problems systematically, we propose a reliability sensitivity and vibration measurement method based on disturbance perception to limit the changing reliability amplitude and frequency. The Fault-Tolerant Cloud Application Reliability Enhancement (FTCARE) strategy using a redundancy mechanism for Microservice-Oriented Cloud Application (MOCA) are proposed to improve QoS. However, the redundant nodes are limited, so distributing the constraint resources to optimize the reliability of cloud applications is a problem answered in this paper.

In this paper, our contributions are as follows:

(i) Firstly, the predicated Petri nets are used to model different components of MOCA, such as microservice request, microservice, microservice composition, and container. Then, we identify the critical microservice in complex cloud applications

(ii) Secondly, a time-homogeneous reliability analysis method based on microservice is proposed. The criticality of each microservice is evaluated through request frequency. The reliability sensitivity is calculated by analyzing the impact of disturbance of the microservice on the whole cloud application. The reliability time series of microservice conforms to the 1st-order Markov Chain evolution rule

(iii) Finally, an active adaptation method to reduce the impact of degradation and fluctuation on microservice composition has been proposed to ensure the reliability of cloud applications. The extensive experiments evaluate the effectiveness of improving MOCA reliability

The rest is organized as follows. First, Section 2 summarizes the existing works. Next, Section 3 and Section 4 introduce the research methodology and modeling cloud application with PrT net, respectively. Then, we propose a ranking algorithm for evaluation and apply the FTCARE strategy in Section 5. Section 6 shows experiments, and Section 7 concludes the paper.

## 2. Related Work

Many types of research have been done in the cloud environment in recent years to improve the availability and reliability of cloud services. Vincenzo and Dragi [5] study and propose a novel method of task unloading with Pareto optimization, which considers three objectives: response time, reliability, and cost. Clab et al. [6] propose a delay adaptive replica synchronization strategy and a replica recovery strategy based on load balancing to solve replica synchronization and recovery of failed nodes in the cloud application. Some indicators have been proposed to measure software reliability, such as Mean Time To Failure (MTTF), Rate of Failure Occurrence (RoCoF), hazard rate, and Probability of Failure on Demand (PoFoD). MTTF and hazard rate are more suitable for system failure with statistical regularity in time. In Ref. [7], the theoretical prediction of the overall MTTF is provided to meet the QoS requirements by fine-tuning the redundancy granularity. Chen et al. [8] improved the cloud service performance by minimizing the propagation of uncertainty in the scheduling workflow application. In Ref. [9], a redundancy minimization algorithm using RIR calculates workflow reliability without calculating the reliability of each task. In order to solve the problems of network instability and limited bandwidth, a reliable VANET routing decision-making scheme based on the Manhattan mobile model is proposed by Gao et al. [10] who consider integrating roadside units (RSU) into wireless and wired modes for data transmission and routing optimization.

A considerable amount of literature focuses on designing new methods of modeling services in the cloud computing environment. Firas and Mazlina [11] offer a literature survey towards agent-based Petri net decision-making modeling for cloud service composition. In Ref. [12], a reliability prediction model based on Petri nets is proposed. For an atomic service, a phased reliability model is proposed to predict the reliability from four aspects: network environment availability, hermit equipment availability, discovery reliability, and critical reliability. A method of constructing reliable service composition is proposed in Ref. [13] with the Petri net, which provides a method to observe the essential behavior of components and describe their relationships. Huang et al. [14] try to solve these challenges by designing a simulation-based optimization method for reliability-aware service composition. A composite stochastic Petri net model of multilayer edge computing system dynamics is proposed and analyzed quantitatively. Zang et al. [15] propose a scheme to automatically generate a service dependency graph using a service registry and improve the service fault tree with a reliability model. In Ref. [16], a Finite State Machine (FSM) model-driven architecture is established, and a typical implementation of the architecture is discussed, including two actual use cases and related evaluations.

Microservices are increasingly regarded as a promising architecture style, building large-scale cloud-based applications within and across organizational boundaries. This microservice-based architecture dramatically improves application scalability, but it also brings expensive performance overhead, which requires careful design of modeling and task scheduling [17]. The proposed methods in this paper are different from the related works that we considered microservice composition with workflow scheduling. Yao et al. [18] propose a fault-tolerant multiprocessor scheduling algorithm using time redundancy. Zhao et al. [19] combine the resubmission and replication with meeting the soft time limit of the workflow. Setlur et al. [20] study the scheduling

problem with less resource redundancy to optimize the reliability and minimize the completion time. In Ref. [21], a replication heuristic algorithm in an unsupervised way has been studied. Safari et al. [22] consider the number of copies, frequency, and reliability and uses the reserved processor to replicate and redistribute copies on the reservation processor.

In the field of cloud-based software stable operation, fault-tolerant technology has been widely concerned with solving the reliability problem under a dynamic and uncertain operating environment. Gao et al. [23] propose a dynamic reconfiguration method of mobile e-commerce service workflow based on cloud edge. Using task replication, Sharif et al. [24] meet the reliability requirements by improving the service quality. Yao et al. [25] propose a nonreplicating model to take over the failed fog node to normalize the fog node by using an operable fog node. Ray et al. [26] propose a novel workflow scheduling algorithm for a cloud application, which combines resubmission and replication. In Ref. [27], an active fault-tolerant system based on CPU temperature is proposed, preempting the federation faults. Shi et al. [28] propose a strategy with mechanism, execution mode, and required resource, ensuring real-time applications. In Ref. [29], the temporal-perturbation aware reliability sensitivity, a disturbance sensing method, is proposed to measure the reliability sensitivity of cloud service components for adaptive cloud service selection.

Although there are many kinds of research on reliability, few current research works consider the reliability of microservice-based cloud application. In particular, we focus on modeling the reliability of cloud applications based on microservice through a redundancy operation. One of the main goals is to reduce the importance of microservices so that the microservices with high request frequency can be backed up. Thus, the reliability of cloud applications is modeled by the fluctuation frequency and amplitude of reliability in a cycle. In this way, the reliability of microservice composition is maximized.

## 3. Framework for Reliability Modeling and Analysis

This section discusses how to model the problem and build the whole framework, covering the reliability analysis approach in the following research. As shown in Figure 1, three blocks with solid lines represent three different modules, and three dotted boxes are the analysis process.

*3.1. User Module.* A user requests the service from the cloud application, in which user requests can be quantitatively described by request frequency. That is, there are $n$ requests from a user per unit time. In each unit time, the failure rate of a microservice is variable due to the change of environment. Besides, in the SaaS (Software-as-a-Service) environment, the microservice composition required by users is usually composed of multiple microservices, which serve with different request frequencies.

*3.2. Microservice Composition Module.* By arranging the microservices in the workflow, the tasks are distributed as a whole cloud application, in which the microservice has an independent failure rate. However, there are dependency and cascade relationships between microservices in practical applications, which will affect the whole application. When a microservice calls the task in the other microservice, the workflow process is regarded as a combination of microservices.

*3.3. Redundancy Module.* The microservices deployed using containers are independent of one another. Each microservice in a container implements multiple tasks such as catalog microservice, ordering microservice, and user profile microservice. An FTCARE strategy with Cloud Application Reliability Optimization (CARO) algorithm for cloud application is proposed to achieve more reliable, more effective and more robust design. Furthermore, the execution model depends on the deadline of microservice and execution types.

*3.4. Reliability Analysis Module.* The formal model of microservice composition with a reliability attribute and its time series is defined. The reliability evaluation method is proposed based on failure probability and exponential reliability equation. The concept of reliability perturbation is defined to describe the reliability changes in continuous time series. The reliability in the evaluation period is calculated with sampled period and failure rate. The microservice sensitivity helps to measure the degradation attribute. The reliability sensitivity based on interference perception and the negative effect of microservice reliability interference is analyzed.

*3.5. Critical Identification Module.* We propose a microservice ranking framework based on the PageRank algorithm for fault-tolerant cloud applications. Since MOCA involves many microservices, it is expensive to provide a redundant replacement for all microservices in cloud applications. Therefore, it is necessary to identify a small number of critical microservices to reduce costs and develop highly reliable cloud applications within a limited budget.

*3.6. Reliability Optimization Module.* To calculate cloud application reliability, we combine the time-homogeneous reliability disturbance analysis to identify the critical microservices. Since the reliability value for each microservice is calculated, the microservices are sorted based on their reliability. The ranking algorithm generates the redundancy scheme. For the FTCARE strategy with redundancies, the most critical microservice determines the optimal solution.

## 4. Reliability Model Based on PrT Net

This section describes the definition of the basic concept and the reliability calculation with predicated Petri net (PrT net), which includes three modeling modules in the framework. Then, the reliability model is analyzed, and the PrT net variables are applied as the parameters in the calculation.

*4.1. PrT Net Syntax and Semantics.* As a graphical modeling tool and formal method with a rich mathematical foundation, PrT net has been widely used to represent the

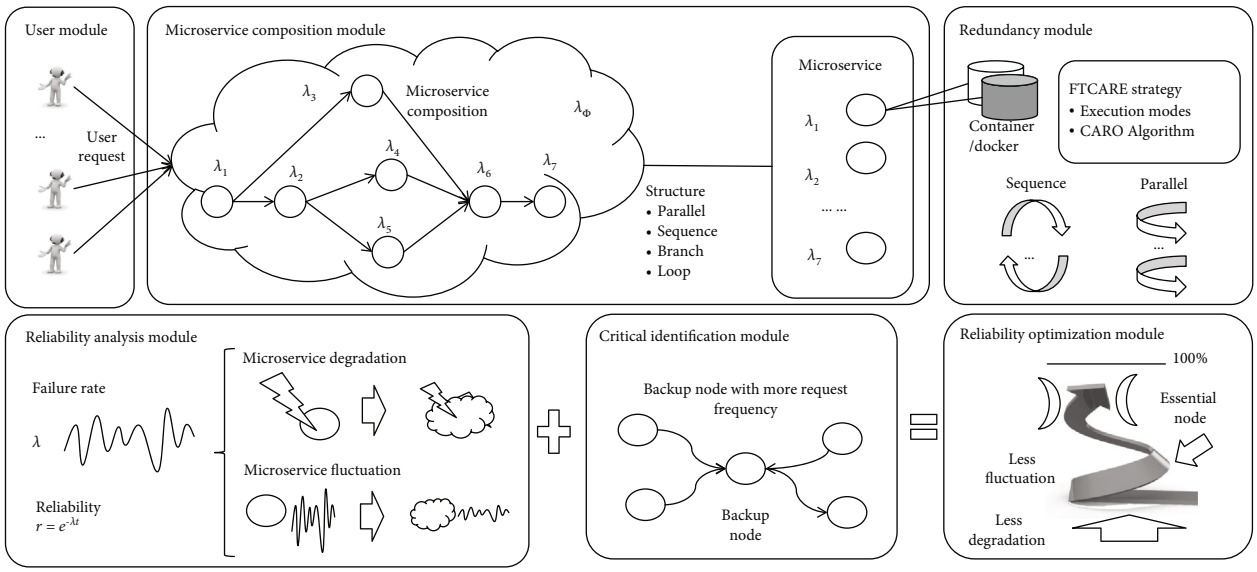



Figure 1: Framework for reliability modeling and analysis.

microservice and influencing factors, and we give the basic definitions as follows:

*Definition 1.* A triple $N = (P, T, F)$ is defined as a Petri net.

(1) $P = \{p_1, p_2, \cdots, p_{|p|}\}$ is a finite set of places

(2) $T = \{t_1, t_2, \cdots, t_{|t|}\}$ is a finite set of transitions and $P \cup T \neq \emptyset$, $P \cap T \neq \emptyset$

(3) $F \subset (P \times T) \cup (T \times P)$ is a set of directed arcs

The PrT net is used to model the execution process of microservice instances. The state, possible operation, and parameter of microservice instances are described by repository, transformation, and interface, respectively. For any $x \in (P \cup T)$, set $x = \{y \mid y \in (P \cup T) \wedge (y, x) \in F\}$ and $x = \{y \mid y \in (P \cup T) \wedge (x, y) \in F\}$ correspond to the input and output of $x$, respectively.

*Definition 2.* A 6-ary tuple $\Sigma = (N, IO, D, A_T, A_F, M_0)$ is called PrT net and used as the primary service net (PSN), where we have the following:

(1) $N = (P, T, F)$ is a basic Petri net, where $P$, $T$, and $F$, denote the finite set of place, transition, and arc, respectively

(2) $IO \subset P$ is a special set, which is the interface of $\Sigma$

(3) $D$ is a nonempty finite individual set, and $f_D$ and $f_S$ are given as predicate sets and symbol sets, respectively

(4) $A_T : T \longrightarrow f_D$, for $t \in T$, the free variable in $A_T(t)$ must be a directed arc free variable with $t$ as one end

(5) $A_F : F \longrightarrow f_S$, if $(p, t) \in F$ or $(t, p) \in F$, then $A_F(p, t)$ or $A_F(t, p)$ is the sum of $n$ary symbols and is null by default

(6) $M : P \longrightarrow f_S$ is the marking of $\Sigma$, $M_0(p)$ is the initial marking

For $\forall t \in T$, if $FV(t_i) = \{x_1, x_2, \cdots, x_n\}$, the individuality set $\{d_1, d_2, \cdots, d_n\}$ meets $d_i \in \{M(p) \mid p \in {}^\bullet t \cup t^\bullet\}$ and $d_i$ corresponds to the variable $x_i$, denoted by $t\langle d_1, d_2, \cdots, d_n\rangle$. For any $t_i \in T$, the input/output arc of transition $t_i$ and the set of free variables in $A_T(t_i)$are denoted as$FV(t_i)$ .$A_T(t)\langle d_1, d_2, \cdots, d_n\rangle$and$A_F(p, t)\langle d_1, d_2, \cdots, d_n\rangle$describe that the individuals $d_1, d_2, \cdots, d_n$ replace the formula $A_T(t)$ and the predicate $A_F(p, t)$, respectively. If $A_T(t)\langle d_1, d_2, \cdots, d_n\rangle$ = true, then $t\langle d_1, d_2, \cdots, d_n\rangle$ is called a feasible replacement of transition $t$ under $M$. All feasible replacements of transition $t$ under $M$ are denoted by set $\mathrm{VP}(M, t)$.

If $\mathrm{VP}(M, t) \neq \emptyset$, then transition $t$ is enabled under $M$, denoted by $M[t >$. Transition $t$ is enabled under $M$ if and only if there is a feasible replacement under $M$. All the enabled transitions under $M$ are denoted by set $\mathrm{ET}(M)$. For transition $t_i \in \mathrm{ET}(M)$, if there is no transition $t_j \in \mathrm{ET}(M)$, whose priority is higher than $t_i$, the firing of transition $t_i$ under $M$ is effective. All the effective firing transitions under $M$ are denoted by set $\mathrm{FT}(M)$. The process that $M$ reaches a new marking $M'$ by firing a feasible replacement $t_i\langle d_1, d_2, \cdots, d_n\rangle$ of transition $t_i$ is denoted by $M[t_i\langle d_1, d_2, \cdots, d_n\rangle > M'$.

We model the microservice MS with PSN, where $P$ is the place in the Petri net and represents the different states of the microservice $\mathrm{MS}(i)$. $T$ is a set of firing microservice, representing the transitions in Petri nets. $F$ is the set of arcs between $p$ and $t$. $W$ is a function defined on $t$, which represents the weight of transitions. $M_0$ represents the initial state of the microservice $\mathrm{MS}(i)$. $M_0 = \{m_{01}, m_{02}, \cdots, m_{0n}\}$, where$m_{0n}$is the number of tokens at state $n$, and $M$ is the number of tokens at $n$.

The microservice composition with many microservices can be defined as the hierarchical framework. On this basis, the HMCN definition for microservice composition with different structures is given.

*Definition 3.* A 6-ary tuple $\Omega = (\Sigma, \Gamma, TI, TA, PI, PA)$ is called hierarchical microservice composition net (HMCN).

(1) $\Sigma$ is a PSN, which describes the basic structure of $\Omega$

(2) $\Gamma = \{\Gamma_i \mid i \in N*\}$ is a finite set of pages, PSN or HMCN. If $N_i = (P_i, T_i, F_i)$, then $\forall \Gamma_i, \Gamma_j \in \Gamma$, $(P_i \cup T_i \cup F)_i \cap (P_j \cup T_j \cup F_j) = \emptyset$

(3) TI $\subset T$ is a collection of alternative nodes, where each page corresponds to a replacement node

(4) TA : TI $\longrightarrow \Gamma$ is the page allocation function, which is used to assign specific pages for a replacement node

(5) PI $\subset P$ is a set of port nodes that describe the input and output locations of the substitute nodes

(6) PA is a port mapping function. The purpose is to map the port node of the replacement node to the interface of the corresponding page

PSN is a special case of HMCN, namely HMCN with $\Gamma$, which is equal to null. HMCN is mainly used for modeling the components in MOCA or the whole microservice composition. Individual set $D$ is mainly used to describe microservice set or available instance set.

According to the microservice relations and behavior characteristics of microservices, PSN and HMCN are mapping with a double circle and a dotted circle, respectively. The page allocation function is to assign specific pages for each replacement node. In order to distinguish between the input interface and output interface, the input interface is marked with superscript $I$, and the output interface is marked with superscript $O$. In particular, only some basic concepts are introduced, and other concepts can refer to Ref. [30].

The microservice request is modeled as HMCN shown in Figure 2, which has the specific operation process: (1) Place $p_s$ is used to store the execution task for request. Transition $t_{\text{req}}$ fires if others request the microservice MS($i$). (2) After obtaining the input parameter $p^I_{\text{ser\_s}}$, the task individual is ready to put into the execution places $p_{\text{exe}}$ and $p_{\text{ctrl}}$ to control the deadline. If the task cannot realize the function within the deadline ($M(p_{\text{to}}) \neq \emptyset$), time-out transition $t_{\text{to}}$ is fired. Otherwise, the execution time is insufficient before task deadline $t > T_{\text{deadline}}$, where $t = T_{\text{arrival}} + t_{\text{execution}}$. The parameters $T_{\text{arrival}}$ and $t_{\text{execution}}$ represent arrival time and consuming execution time in the container, respectively. Next, the task results are fed back from the container ($M(p^O_{\text{containero}}) \neq \emptyset$), then output after firing $t_{\text{exe}}$ is transferred to place $p^O_{\text{ser\_e}}$. Finally, place $p_e$ is the termination of the microservice request.

Places $p^I_{\text{ser\_s}}$ and $p^O_{\text{ser\_e}}$ with dotted circles are mapping to places $p_{\text{ser\_s}(i)}$ or $p_{\text{ser\_s}(j)}$ and $p_{\text{ser\_e}(i)}$ or $p_{\text{ser\_e}(j)}$ with double circles, respectively, which define the fundamental structures of microservice composition in Definition 4. Places $p^I_{\text{containeri}}$ and $p^O_{\text{containero}}$ with dotted circles are mapping to places $p_{\text{containeri}}$ and $p_{\text{containero}}$ with double circles, respectively, which formulates the redundancy modes in Definition 5.

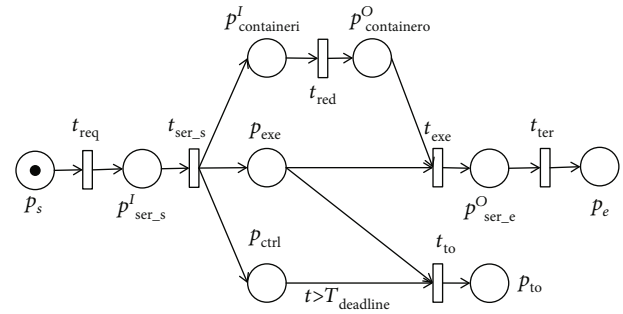



Figure 2: Modeling microservice request.

*Property 1.* The state of request service model can be either executed in the container or out of time.

*Proof.* Given a token as user request in place $p_{\text{exe}}$, $M(p_{\text{exe}}) \neq \emptyset$. When the condition $t > T_{\text{deadline}}$ satisfies, then transition $t_{\text{to}}$ is fired, $M(p_{\text{exe}}) = \emptyset$. The prepositional place $p_{\text{exe}}$ of transition $t_{\text{to}}$ has no token anymore, so the token cannot reach place $p_{\text{e}}$. On the contrary, it can be proved too. □ □

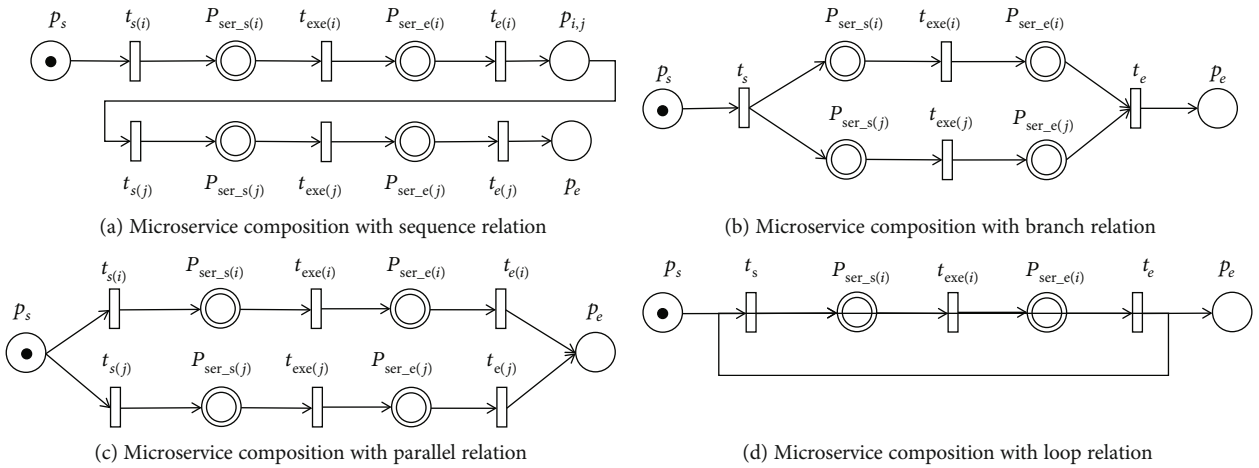
In this HMCN model, places $p^O_{\text{ser\_e}}$ and place $p_{\text{to}}$ represent two states (executed in container successfully and running out of time) in the microservice request model, respectively. They are proved to be mutually exclusive.

*Definition 4* (microservice relation). The symbols MS($i$) > MS($j$), MS($i$) + MS($j$), MS($i$) ∥ MS($j$), and MS($i$)◎MS($j$), represent the sequence, branch, parallel, and loop relations between MS($i$) and MS($j$), respectively.

The microservice relation of microservice composition is modeled with RL($i$, $j$), which represents the relation between MS($i$) and MS($j$). RL($i$, $j$): MS($i$) × MS($j$) $\longrightarrow$ {>,+,∥,◎} is the relationship between tasks, as depicted in Figure 3.

*Definition 5* (redundant fault-tolerant mechanism). A pair RFTM = (FT, MODE) is used to describe the redundant mechanism for fault tolerance. The redundant microservice is assigned in light of the CARO algorithm. FT(MS($i$)) = {MS($i$, 1), MS($i$, 2), ⋯, MS($i$, $r$), ⋯, MS($i$, $R$)}. MS($i$) is the primary microservice, and MS($i$, $r$) is the $r$th backup of microservice MS($i$). When the primary microservice MS($i$) fails, the standby microservices MS($i$, $r$) starts, thus improving the reliability. MODE = {SEQ, PAR}. Under the deadline of the microservice $T_{\text{deadline}}$, the redundancy mode can be sequence or parallel, in which the backups are executed actively or passively. For active backup execution, the primary and secondary copies are executed at the same time. When the primary copy runs successfully, the secondary copy is terminated. For passive backup execution, after the primary copy fails, the secondary copy starts to run. If the task execution is complete, the container is released.

The container model realizes the redundant fault-tolerant mechanism, as shown in Figure 4. Place $P_{\text{containeri}}$ stores the queuing tasks of the container as tokens. If the



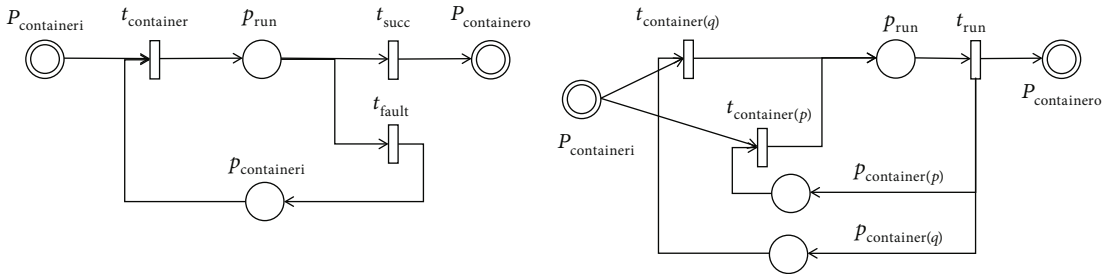

(a) Microservice composition with sequence relation

(b) Microservice composition with branch relation

(c) Microservice composition with parallel relation

(d) Microservice composition with loop relation

Figure 3: Modeling basic structures of microservice composition.




(a) Parallel mode

(b) Sequence mode

Figure 4: Modeling container with two modes.

processing capacity of the container is faster than the request frequency, then transition $t_{container}$ is fired. Finally, place $p_{run}$ represents the execution state of tasks.

*Property 2.* The designed PrT net can effectively describe the microservice backup redundancy in the container.

*Proof.* According to the backup execution mode, there are two redundancy modes: sequential and parallel. Therefore, this property is proven from the following two parts:

(i) When the condition $(T_{deadline} - T_{arrival}) > 2 * t_{execution}$ is satisfied, then the sequence redundancy mode works. In case of failure, the transition $t_{fault}$ fires, and the token is transited to $p_{container}$, which is ready for the next iteration

(ii) When the condition $(T_{deadline} - T_{arrival}) < 2 * t_{execution}$ is satisfied, then the parallel mode is selected for redundancy. The synchronized execution of microservices is realized by $t_{run}$. Transitions $t_{container(p)}$ and $t_{container(q)}$ fire independently, representing the parallel redundancy mode

$T_{deadline}$ is a given variable, and $T_{arrival}$ depends on the actual deployment of the container and the schema of the microservice composition. ☐ ☐

The microservice MS in microservice composition MC performs on specific nodes and links. In case of failure from the node or link, the whole composition will also be affected. The reliability of microservice composition is analyzed based on the PrT net. The reliability requirement for MOCA is defined below.

*Definition 6* (cloud application reliability requirement). The reliability of cloud application is a five-tuple (MS, MC, RD, RP, $V$). MS and MC are the microservice and the microservice compositions defined above, respectively. For the microservices $\{MS(1), MS(2), \cdots, MS(i), \cdots, MS(n)\}$, the parameters $RD = \{rd_{(1,MC)}, rd_{(2,MC)}, \cdots, rd_{(n,MC)}\}$ and $RP = \{rp_{(1,MC)}, rp_{(2,MC)}, \cdots, rp_{(n,MC)}\}$ represent reliability degradation and perturbation, which are two basic reliability indicators for the measurement. The criticality value $V = \{V_1, V_2, \cdots, V_n\}$ of each microservice can be calculated with microservice relation and request frequency.

In this paper, RD and RP are the microservice reliability quality parameters analyzed in Section 5 and the criticality value $V$ helps to measure the role of a microservice.

*4.2. Reliability Calculation.* Each microservice has a special attribute failure rate, which describes the number of failures over a while. The failure rate can be defined as the expected number of project failures in a specified period [31]. The

failure rate of a microservice conforms to the normal distribution $N(\mu, \sigma^2)$ in this paper.

In the exponential reliability calculation, the failure rate parameter (i.e., $\lambda_i$ for the $i$th microservice) determines the reliability evaluation indicator and time $t$ is measured as the execution time of a microservice.

$$r_i = e^{-\lambda_i * t}. \tag{1}$$

Let MC be the microservice composition, and $\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$ be the failure rate of $n$ microservices. The failure rate of the microservice composition is calculated based on the workflow execution. MOCA may contain more than one microservice relation. Wang et al. [29, 30] provides the aggregation functions in Table 1 to calculate the $\lambda_{\mathrm{MC}}$ of the sequential, parallel, branching, and loop relations of microservice composition.

The parameter $b_i$ represents the execution probability of the $i$th branch in microservice composition, and $l_i$ is used to represent the execution probability of $i$ times in a cycle.

Due to the uncertainty of the internal working state and the instability of the cloud operation environment, the time interval of fault occurrence is uncertain. For a specific microservice or the microservice composition in cloud application within a specific time interval $\Delta T$ in a lifetime, the reliability is as follows:

$$r^{\Delta T} = e^{-\lambda * \Delta T}. \tag{2}$$

Unstable links in microservice composition will affect the response time of microservices in the lifetime intervals $\Delta T(0), \Delta T(2), \cdots, \Delta T(m), \cdots, \Delta T(M-1)$, which may lead to execution out of time or unexpected failure during processing tasks. $\Delta T(m)$ represents the $m$th period $\Delta T$ in the time series. Each period has a reliability evaluation value $r^{\Delta T(m)}$.

According to Ref. [32], the evolution in reliability time series satisfies the 1st-order Markov Chain rule. Moreover, since uncertain events trigger a single one-step transition, the statistics of multiple one-step transitions in advance can reflect the long-term evolution law of reliability.

## 5. Reliability Analysis and Algorithm

This section analyzes the cloud application reliability and proposes a redundancy mechanism, three analysis modules in the framework.

*5.1. Reliability Perturbation Analysis.* This section describes the dynamic influence of microservice perturbation on microservice composition.

The microservice is running under a specific status during a period. For any $m \in [1, M-1]$, the $m$th reliability perturbation of the $i$th microservice within $\Delta T(m)$ can be defined as follows:

$$p_i^{\Delta T(m)} = r_i^{\Delta T(m+1)} - r_i^{\Delta T(m)}, \tag{3}$$

Table 1: The calculation for the failure rate of microservice composition according to structure.

| Structure | Failure rate calculation |
|---|---|
| Sequence | $\lambda_{\mathrm{MC}} = 1 - \prod_{i=1}^{n}(1-\lambda_i)$ |
| Parallel | $\lambda_{\mathrm{MC}} = 1 - \prod_{i=1}^{n}(1-\lambda_i)$ |
| Branch | $\lambda_{\mathrm{MC}} = 1 - \prod_{i=1}^{n} b_i * (1-\lambda_i),\ \sum_{i=1}^{n} b_i = 1$ |
| Loop | $\lambda_{\mathrm{MC}} = 1 - \prod_{i=1}^{n} l_i * (1-\lambda_i)^i,\ \sum_{i=1}^{n} l_i = 1$ |

where $r_i^{\Delta T(m)}$ represents the reliability of microservice $s_i$ within $\Delta T(m)$. Similarly, $r_{\mathrm{MC}}^{\Delta T}$ represents the reliability of microservice composition MC within $\Delta T(m)$.

In order to eliminate the incompatibility between the different parameters, the Min-Max normalization technique widely used in formula (4) is used to normalize the original value. After performing Min-Max normalization for perturbations, the reliability perturbation rate of the $i$th microservice within $\Delta T(m)$ is defined as follows:

$$pr_i^{\Delta T(m)} = \begin{cases} \dfrac{p_i^{\max} - p^{\Delta T(m)}}{p_i^{\max} - p_i^{\min}}, & \text{if } p_i^{\max} \neq p_i^{\min}, \\ 1, & \text{if } p_i^{\max} = p_i^{\min}, \end{cases} \tag{4}$$

where

$$p_i^{\max} = \max_{m \in [1, M-1]} p_i^{\Delta T(m)}, \tag{5}$$

$$p_i^{\min} = \min_{m \in [1, M-1]} p_i^{\Delta T(m)}. \tag{6}$$

Similar to formula (3) and formula (4), $p_{\mathrm{MC}}^{\Delta T(m)}$ and $pr_{\mathrm{MC}}^{\Delta T(m)}$ are perturbation and perturbation rate of microservice composition MC within $\Delta T(m)$.

The perturbation function for the single-step transition can be regarded as a 1st-order Markov Chain evolution, which describes the evolution rules of the latest reliability time series of a microservice. The multiple times of perturbations in the 1st-order Markov evolutionary reliability time series can reflect the cumulative effects. The closer the disturbance is, the more accurate the execution state is.

A reliability sensitivity measurement method based on perturbation perception is proposed based on reliability perturbation, and the cumulative negative impact of perturbation is analyzed. Assuming $(M-1)$ times of perturbations in time series, the cumulative effect defines an influencing factor parameter $\alpha \in [0, 1]$ and the influencing factor for the $m$th perturbation is $\alpha^{M-m}$. Through calculating the reliability sensitivity, the perturbation based on the disturbance is measured by

$$rp_{(i,\mathrm{MC})} = \frac{1}{M-1} * \sum_{m=1}^{M-1} \alpha^{M-m} \pi^{\Delta T(m)}, \tag{7}$$

where

$$\pi^{\Delta T(m)} = \begin{cases} \dfrac{pr_{\mathrm{MC}}^{\Delta T(m)}}{pr_i^{\Delta T(m)}}, p_{\mathrm{MC}}^{\Delta T(m)} > 0, \\ 0, p_{\mathrm{MC}}^{\Delta T(m)} \le 0. \end{cases} \tag{8}$$

The reliability sensitivity $\pi^{\Delta T(m)}$ in formula (7) is equal to zero when the perturbation of microservice composition MC within $\Delta T(m)$ is negative. The more significant the positive impact on the composition of microservices, the higher the sensitivity of microservices. If the microservice has a negative effect, the sensitivity is 0.

*5.2. Reliability Degradation Analysis.* The degradation caused by microservice runtime exceptions may lead to cascading effects in cloud applications. This section describes the static impact of microservice degradation on microservice composition.

In the dynamic cloud environment, the changes lead to the upgrading or degradation of a microservice, measured by microservice sensitivity.

The fluctuation rate of microservice $s_i$ at continuous time series points is calculated with formula (9):

$$fr_{(i,\mathrm{MC})} = \frac{1}{M-1} * \sum_{m=0}^{M-1} \pi^{\Delta T(m)}, \tag{9}$$

where

$$\pi^{\Delta T(m)} = \frac{pr_{\mathrm{MC}}^{\Delta T(m)}}{pr_i^{\Delta T(m)}}. \tag{10}$$

The reliability fluctuation rate with $M$ time intervals $\Delta T$ is represented by the ratio of the reliability of microservice composition to the reliability of microservice $s_i$ in unit time.

The reliability degradation of microservice composition MC under influence of microservice $s_i$ is calculated as follows:

$$rd_{(i,\mathrm{MC})} = \frac{fr_{(i,\mathrm{MC})}}{\sum_{i=1}^{n} fr_{(i,\mathrm{MC})}}. \tag{11}$$

*5.3. Critical Microservice Identification.* This section describes the importance of user requests to each microservice from the structure level of microservice composition.

In the cloud application, some microservices are often requested by others. These microservices are considered more critical because their failure has more impact on the microservice composition than other normal ones. Intuitively, critical elements in MOCA are those that have many requests from other critical ones. Inspired by the PageRank algorithm in Ref. [33, 34], we propose an algorithm to measure the criticality of cloud microservices.

A microservice composition can be modeled as a weighted directed graph G, where a node $s_i$ in the graph represents a microservice and a directed edge $e_{j,i}$ from node $s_i$ to node $s_j$ represents the invocation relationship, i.e., $s_i$ invokes $s_j$. Each node $s_i$ in graph G has a nonnegative criticality value $V_i$, which is in the range of $(0, 1)$. Each edge $e_{j,i}$ in the graph has a nonnegative weight value $W(e_{j,i})$, which is in the range of $[0, 1]$. The weight value of an edge $e_{j,i}$ based on request frequency can be calculated by

$$W(e_{j,i}) = \frac{\mathrm{freq}_{j,i}}{\sum_{j=0}^{p} \mathrm{freq}_{j,i}}, \tag{12}$$

where $\mathrm{freq}_{j,i}$ is the invocation frequency of microservice $s_j$ by other microservice $s_i$ and $p$ is the number of microservices invoked by $s_i$. In this way, the edge $e_{j,i}$ has a more considerable weight value if microservice $s_j$ is invoked more frequently by microservice $s_i$ compared with other microservices invoked by $s_i$.

The measurement of the microservice importance is based on invocation relationship and request frequency [35]. The microservices are considered more critical if many other essential microservices frequently request them. This microservice failure has more impact on the whole cloud application than the normal microservices. The criticality value for the microservice $s_i$ is as follows:

$$V_i = \frac{1-d}{n} + d * \sum_{j=1}^{p} V_i * W(e_{j,i}), \tag{13}$$

where $n$ is the number of microservices and $p$ is the number of microservices that invoke microservice $s_i$. The parameter $d(0 \le \mathrm{d} \le 1)$ in equation (13) is employed to adjust the criticality values derived from others so that $V_i$ is composed of the fundamental value of itself (i.e., $(1-d)/n$) and the derived values from the microservices that invoked $s_i$.

By formula (13), microservice $s_i$ has a more considerable criticality value if the number of predecessors microservices $p$ and their weight value $W(e_{j,i})$ are large, indicating that microservice $s_i$ is invoked by the other critical microservices frequently.

Reliability degradation measures the degree of microservice composition affected when microservice is degraded. The critical microservice identification indicates the importance of a microservice in the microservice composition with request frequency.

*5.4. CARO Algorithm.* This section will analyze the reliability of cloud applications and explain the CARO algorithm based on redundancy technology.

*Input:* $\lambda_i^{\Delta T(m)}$, $\gamma$, number of redundant microservices $R$, microservice composition schema $S$, and $M$ time series from initial time interval $\Delta T(0)$ to final time interval $\Delta T(M-1)$ number of microservices $n$
*Output:* index list of redundant microservices *ReS*
1 calculate $r_i^{\Delta T(m)}$ with formula (2);
2 calculate $\lambda_{MC}^{\Delta T(m)}$ and $r_{MC}^{\Delta T(m)}$ with formula (2) and formulas in Table I;
3 *for* $r = 1$ to $R$ *do*
4 *for* $i = 1$ to $n$ *do*
5 calculate $rd_{(i,MC)}$ with formula (7);
6 calculate $rp_{(i,MC)}$ with formula (11);
7 calculate $V_i$ with formula (15);
8 *if* $r_{MC} \le r_{MC}^E$ *then*
9 calculate $CAR_{(i,MC)}$ with formula (11) and formula (15);
10 *else*
11 calculate $CAR_{(i,MC)}$ with formula (7);
12 ranking $CAR_{(i,MC)}$ and get max index *i*;
13 add index *i* to *ReS*;
14 update microservice composition schema *S*;
15 return *ReS*;

Algorithm 1: CARO.

The microservice fails only if all the backups fail, so formula (14) can calculate the failure rate after $R$ backups the redundancy process:

$$\lambda_i^R = 1 - \prod_{r=1}^{R} (1 - \lambda_i^r). \tag{14}$$

Let $R$ be the number of redundant microservices and $r_i^r$ be the reliability of the $r$th redundancy of microservice $s_i$.

The criticality value is updated because of changing the microservice schema.

$$V'_i = \frac{1-d}{n} + d * \sum_{j=1}^{p'} V'_i * W(e_{j,i}), \tag{15}$$

where

$$p' = p + |s_i^r|. \tag{16}$$

In formula (16), $|s_i^r|$ is the redundancy number of microservice $s_i$ and $p'$ is the number of microservices that invoke microservice $s_i$ after redundancies.

To measure the negative influences of microservice reliability degradation and perturbation, we combine the degradation and perturbation, considering microservice criticality, and present a novel approach for cloud application reliability measurement. The cloud application reliability of microservice composition MC under the influence of microservice $s_i$ is calculated as follows:

$$\mathrm{CAR}_{(i,\mathrm{MC})} = \begin{cases} \gamma * rd_{(i,\mathrm{MC})} + (1-\gamma) * V'_i, & r_{\mathrm{MC}} < r_{\mathrm{MC}}^E, \\ rp_{(i,\mathrm{MC})}, & r_{\mathrm{MC}} \ge r_{\mathrm{MC}}^E. \end{cases} \tag{17}$$

In the case of lower reliability, $r_{\mathrm{MC}} < r_{\mathrm{MC}}^E$ in formula (17), the purpose is to improve the microservice reliability and reduce its degradation to the cloud application. $r_{\mathrm{MC}}^E$ is the expected value of reliability for microservice composition MC. After achieving the desired value, the main objective is to stabilize the perturbation, and $rp_{(i,\mathrm{MC})}$ measures the reliability quality dominantly.

The CARO algorithm includes two parts: (1) ranking the influence degree from microservice reliability to cloud application and (2) selecting the optimal fault-tolerant strategy. The development procedures are as follows:

(1) The initial architecture design of MOCA is provided with a microservice composition graph. The microservices can be sorted based on the cloud application reliability, and the most critical microservices can be identified. The ranking results determine the allocation of redundancies (line 12 in Algorithm 1).

(2) Since the most critical microservices are identified, each candidate performance is calculated, and the most suitable redundancy resource is selected for each important microservice (lines 5-11 in Algorithm 1) after iteration. The improved design and ranking results are updated to the redundancy scheme again (line 14 in Algorithm 1) after iteration

The output value *ReS* in the CARO algorithm represents the redundancies for each microservice in the schema. After

the reliability optimization process, we can obtain the updated microservice composition schema with redundancies.

## 6. Simulation

We conducted a series of experiments to evaluate the effectiveness of the proposed FTCARE strategy. This paper focuses on the reliability guarantee of cloud applications based on a microservice. In conclusion, the experiment is aimed at answering the following Research Questions (RQ):

(i) Proving the effectiveness of the FTCARE strategy with different workflow examples. According to the analysis, the cloud application reliability based on a microservice increases with redundancy operation, and minor perturbation is expected

(ii) Comparing the cloud application reliability with and without considering degradation. Considering the degradation, the improvement of microservice reliability is faster for saving time to stabilize reliability perturbation

*6.1. Simulation Environment.* The reliability improvement approach is implemented in Java with Python 3.6.0 and PyCharm 2020.1.2. A program package for analysis and visualization of cloud application reliability is used to simulate microservice composition workflows. Microservice reliability data sets are obtained from papers of other researchers, and other data are generated in simulation.

Take ten workflows in Ref. [36] as examples, as depicted in Figure 5. These ten commonly used workflow models are based on the basic structure of microservice composition, including serial and parallel. By mixing the basic microservice composition structure, we can get individual examples. In ten groups of workflows, each microservice has an independent failure rate. The microservice composition composed of microservices also has its failure rate and reliability. The failure rate of microservice composition is determined according to the microservice relations modeling and analysis.

Fourteen microservices are selected as the essential simulation elements, which run for a period of 100 s. In order to simplify the actual scene, in the simulation experiment, each workflow node only runs one microservice. That is, each task is mapping one microservice. During the runtime, the failure rate varies in the range of $(0, 0.1]$, and we observe 99 intervals for simulation. The microservice failure rate dataset is generated randomly because the failure in an actual situation occurs by accident or under rapidly changing conditions. Moreover, in the observed execution time, the microservice failure rate conforms to a normal distribution in formula (18) with a mathematical expectation of $\mu = 0.05$ and variance $\delta = 0.015$.

$$\lambda_i^{\Delta T(m)} = \text{random} \bullet \text{normal}(\mu, \delta, N), \tag{18}$$

where $m$ represents the time interval [1,100] and $i \in [1, 14]$. $N$ is the sample number.

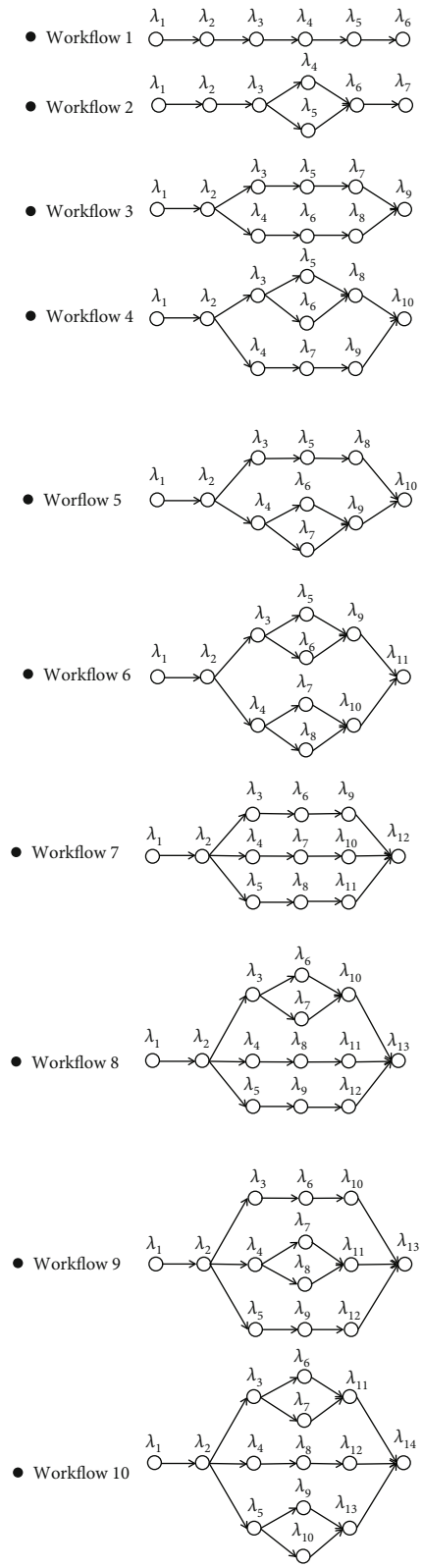



Figure 5: Ten workflows of microservice composition for simulation.

The request frequency between its processor or successor is generated randomly in the interval [80,100] with formula (19).

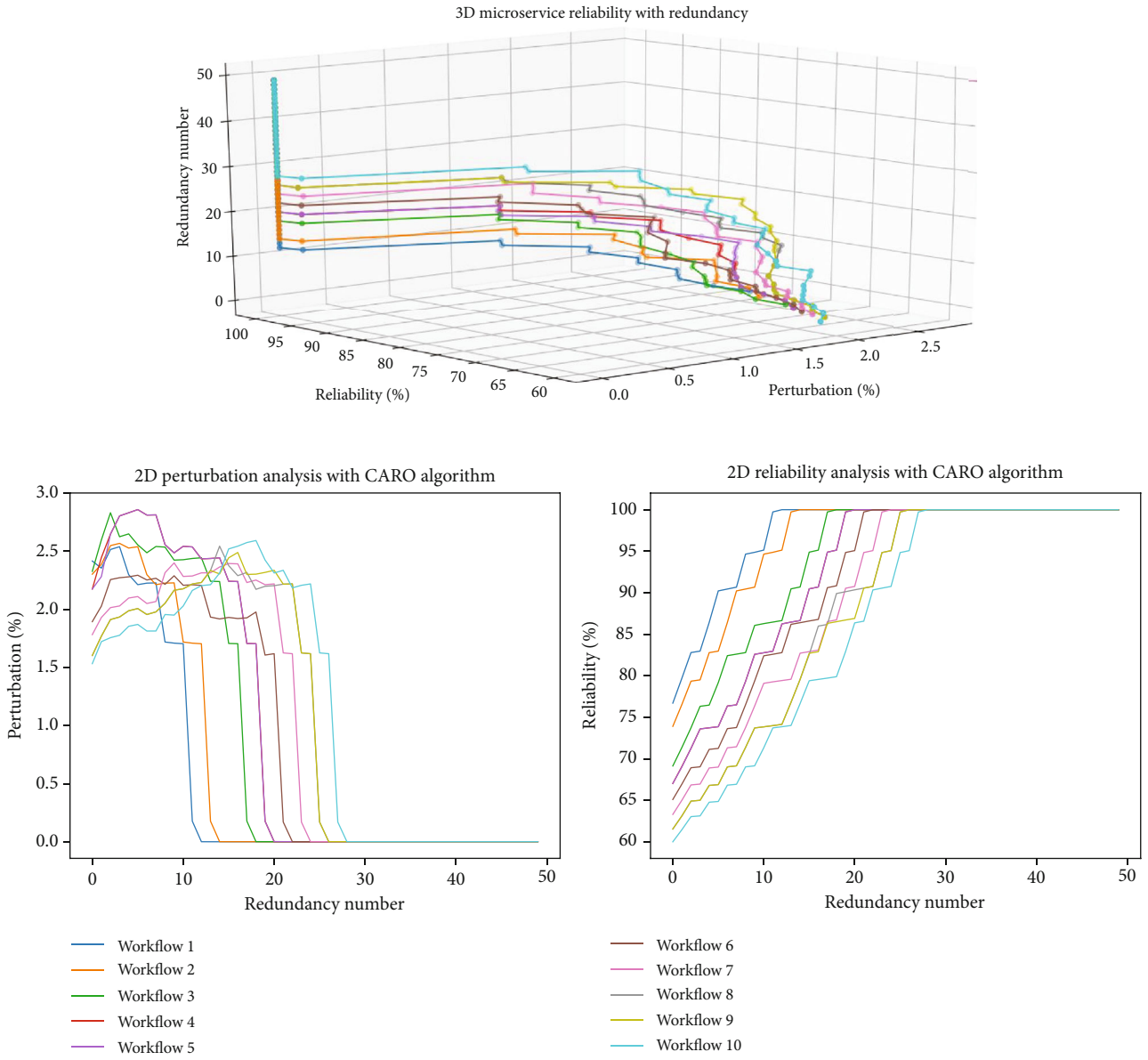



Figure 6: Reliability analysis for ten workflows.

$$\text{freq}_{j,i} = \text{random}() * \text{scale} + \text{min}, \tag{19}$$

where scale = 20 and min = 80.

The purpose of the experiment is to explore the reliability of microservice composition under dynamic disturbance. In order to simplify the simulated experiment, the request frequency is constant during the whole execution time in the experiment.

The redundancy scheme is designed as an array with repeating elements, representing the redundant microservice, and the failure rate will update by replicating critical elements in Section 5. For example, the scheme array [1, 3, 5] means the first microservice has two redundancies, and the third and fifth microservice has only one backup.

The details about the sequence and parallel modes are not considered in simulation related to the execution time and subdeadline of microservices. Instead, we only consider the redundancy number, and the execution mechanism is viewed as a black box.

*6.2. Simulation Results.* Through the simulation of ten data groups, we can calculate the reliability with redundancies quantitatively. Figure 6 shows the analysis of cloud application reliability after simulating ten workflows. The observed variables are reliability and disturbance. We can find that the algorithm proposed in this paper helps improve the QoS performance of microservice composition based on the fluctuation of existing microservice reliability through these ten sets of data. The trend shows that with the increase of

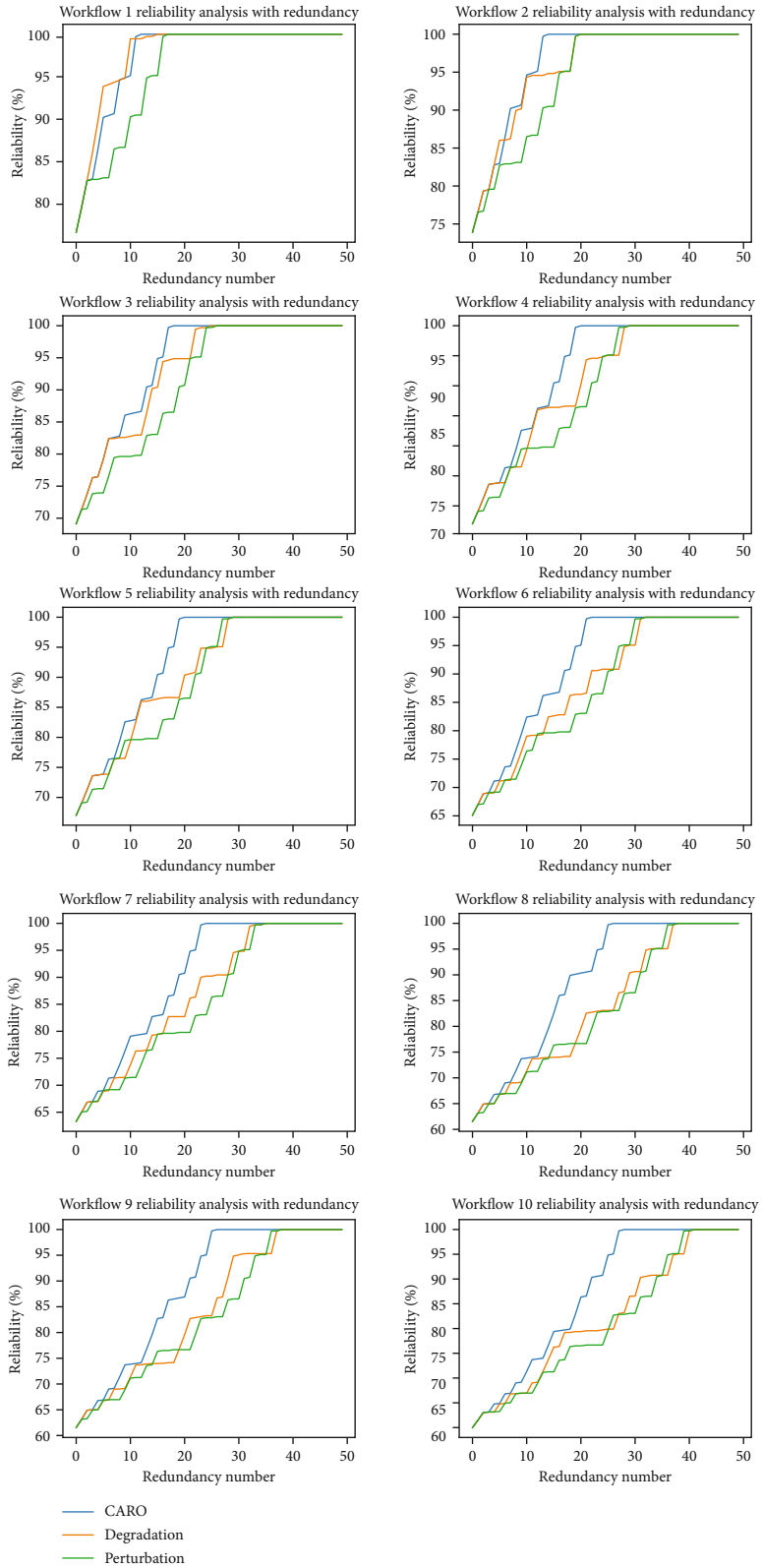



Figure 7: Reliability analysis with various algorithms for 10 workflows.

redundancies, the reliability of microservice composition improves, and its vibration decreases.

As shown in Figure 6, the reliability disturbance decreases from 6% to nearly 0% as redundancies increase. However, this change is not linear because the disturbance is focused on the difference in microservice reliability between adjacent sampling periods in time series. Thus, by improving the reliability of microservices, the reliability of

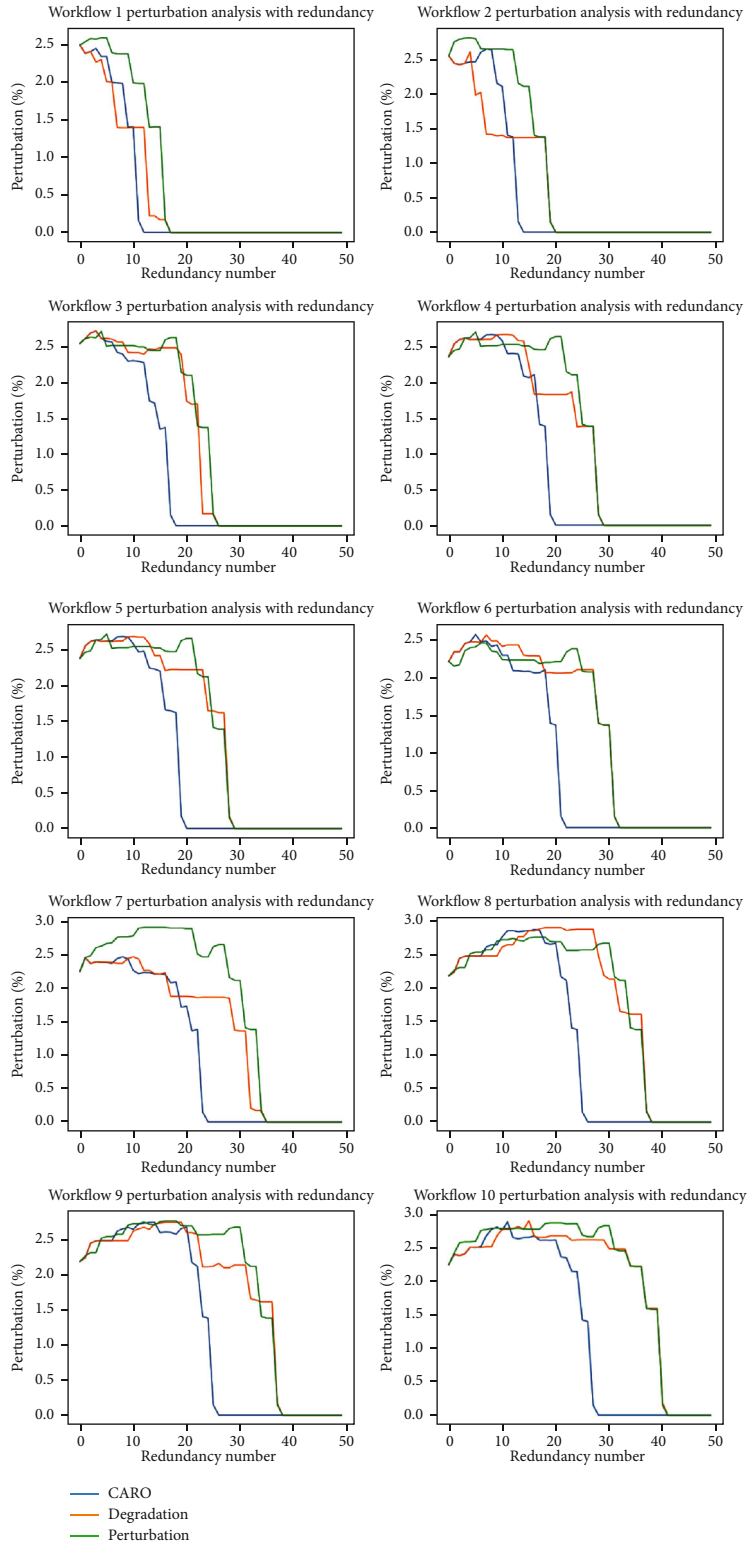



Figure 8: Perturbation analysis with various algorithms for ten workflows.

microservice composition is also improved. However, the ratio between them is uncertain in the short term considering the influence of timing.

Besides, our algorithm is designed to discuss the reliability fluctuation under the premise of high reliability. Therefore, it does not make sense if the microservice composition remains stable at low reliability. Therefore, with the improvement of redundancy operation, it is expected that the overall reliability will be improved, and the fluctuation will tend to be stable.

*6.3. Algorithm Comparison and Result Analysis.* The emphasis of this section is to compare the influence of different redundancy strategies. We use the CARO algorithm without considering degradation and perturbation, selecting three algorithms for comparison by simulating ten workflows.

The algorithm without considering degradation is only for microservice disturbance redundancy. Therefore, we optimize reliability, as shown in the high-reliability condition. In Figure 7, the redundancy helps improve reliability, but it will take a long time to achieve high reliability if only for the fluctuation of the redundancy operation. The iterative steps will be wasted on maintaining the stability of microservice composition and ignoring the reliability parameters.

Figure 7 shows that the proposed CARO algorithm performs better than the others. The microservice composition reliability can achieve a higher value because of the ranking process. For each iteration, the microservices with the top reliability value are replicated. To improve the effectiveness of the FTCARE strategy, we distinguish the levels so that the application reliability is not confused with perturbation at a low level.

The CARO algorithm with ranking selection performs not as well as the other two algorithms but can also achieve high reliability with minor vibration by increasing the redundancy number finally.

Comparing our ten sets of trials, each set of data fits the rules of the above analysis. Hence, there is a wide range of applicability, not for an individual microservice composition. From the diagram data, we can see that as the number of microservices increases and their complexity increases, it takes longer to reach an expected state.

Figure 8 compares the CARO algorithm perturbation performance with degradation and perturbation analysis for workflow simulations. Again, the CARO algorithm can reach the lowest perturbation much faster than the other two.

The reliability perturbation reduces not continuously because it offers dynamic vibration during execution time. For example, the reliability of the $i$th microservice at $\Delta T(m-1)$, $\Delta T(m)$, and $\Delta T(m+1)$ time intervals are 60%, 60%, and 65%, and the maximum perturbation is 5%. However, in the redundancy process, the reliability is 90%, 80%, and 95%, and the perturbation is 10%, which means the reliability perturbation performs terribly at this time point. However, with the improvement of reliability, the vibration space is narrower. For example, if overall reliability achieves more than 95%, the maximum reliability perturbation is only 5% because it cannot exceed 100% in reality.

Besides that, we can find that the impact of redundancy optimization strategy on stability is more prominent with the growth of the number of microservices and the improvement of schema complexity. For workflow 10, the earlier stability of microservice composition is achieved by using fewer redundancies, and the optimization efficiency is much higher. In workflow 1, the efficiency difference of the three algorithms is not particularly obvious. As the number of microservices increases, the amount of redundancies required also increases to achieve the stability goal.

Similarly, ten trials found that, as the number of microservices and the complexity of microservice composition structure increases, it takes longer to reach a stable state. In other words, with the improvement of workflow complexity, the efficiency of our optimization CARO algorithm is higher.

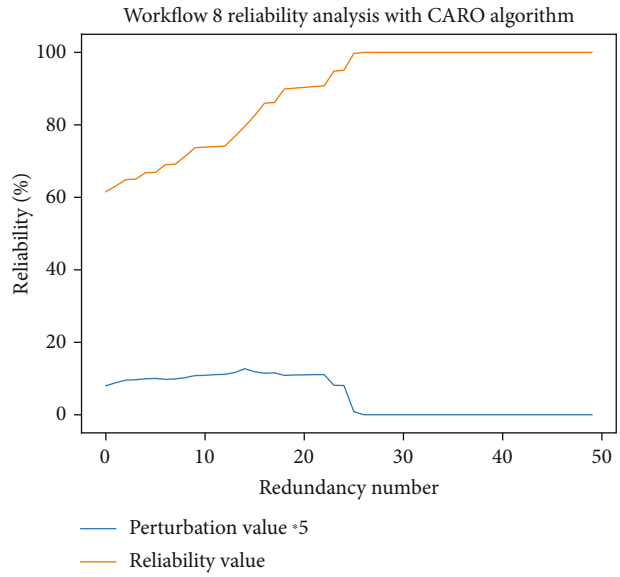



Figure 9: Analysis of the expected reliability $r^E_{MC}$ for workflow 8.

Our proposed CARO algorithm takes high reliability as a priority and fluctuation as the second optimization factor. Therefore, the fluctuation is relatively large in the early stage of the optimization disturbance process, reaching an expected state. In contrast, without degradation, which takes the influence of optimization fluctuation as the only factor, it has high efficiency of disturbance optimization but may produce low reliability for a while.

As we analyzed, in low-reliability, degradation ranking is used to update the redundancy scheme. Thus, the redundancy operation improves the microservice composition reliability on one side. On the other side, it reduces or narrows the degradation. Both effects help speed up the optimization effectiveness, and after that, perturbation plays a role instead.

*6.4. Parameter Analysis.* We have used some variables in the calculation formula to analyze the reliability of cloud applications based on microservices in Section 5. In the following part, we will discuss the influence of these variables on reliability optimization.

CARO optimization focuses on steady improvement by computing the reliability of cloud applications. We have defined the variables of expected reliability $r^E_{MC}$. When the reliability is lower than the threshold $r_{MC} < r^E_{MC}$, the main objective of the optimization is to improve the reliability of the application. When the reliability reaches the threshold $r_{MC} \geq r^E_{MC}$, the purpose of optimization is to eliminate the fluctuation. Therefore, we take workflow eight as an example to verify whether the optimization goal is achieved. In order to make the comparison more apparent, the fluctuation of reliability is expressed by multiplying its value by five. In Figure 9, the threshold of cloud applications is 80%. When the reliability value is

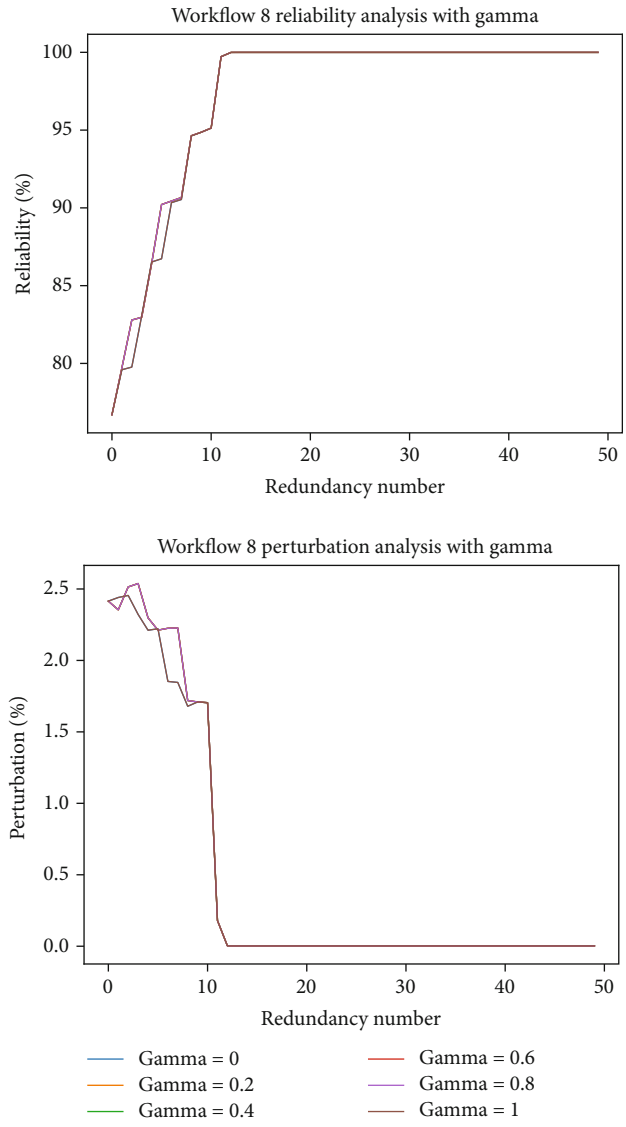



Figure 10: Reliability and perturbation analysis of parameter $\gamma$ for workflow 8.

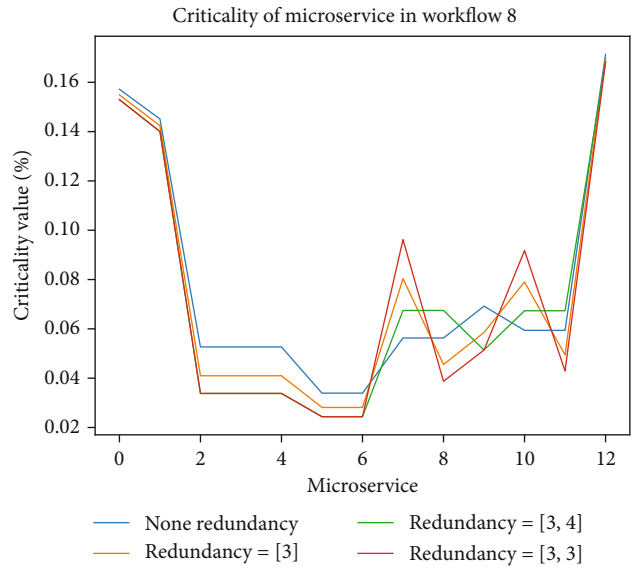



Figure 11: Experimental result with critical value with and without redundancy.

lower than it, the overall reliability is improved, but the fluctuation is relatively high. When the reliability reaches 80%, the overall reliability is improved, but the fluctuation also tends to be gentle.

We can find that other workflow mentioned in this paper can also meet this optimization goal through our experimental comparison. Similarly, when we modify the expected value of reliability $r^E_{\mathrm{MC}}$, the figures of reliability analysis follow the same rule. However, because of the length of the article, we do not demonstrate all the pictures here. In the practical MOCA applications, both positive and negative fluctuations of reliability are not expected. However, the positive impact of microservice reliability on service composition is pursued, while the negative impact is suppressed through a reliability improvement strategy. That is where we optimize the difference between reliability degradation and fluctuation. The goal of fluctuation is to be stable, and the goal of degradation is to enhance.

When computing the reliability of cloud services, the parameter $\gamma$ divides the importance and degradation of microservices. With experience, we take $\gamma$ as a number from 0 to 1. On the one hand, we need to consider the importance of a microservice in microservice composition schema. On the other hand, the impact of microservice reliability on microservice composition also needs to be studied. Therefore, the value of $\gamma$ needs to be discussed.

In order to analyze the influence of $\gamma$ on reliability optimization, we use workflow eight as a simulation example. When $\gamma = 1$, the impact of reliability degradation on microservice composition is considered first. When $\gamma = 0$, the criticality of a microservice in microservice composition plays a decisive role. The value of $\gamma$ between 0 and 1 indicates the trade-off between the both indicators.

We can see from Figure 10 that the value of $\gamma$ does not affect the reliability and fluctuation. In other words, if reliability degradation has a significant impact on microservice composition, it is equally essential in the schema. Thus, we can use parameters $rd_{(i,\mathrm{MC})}$ and $V'_i$ to analyze the reliability characteristics when the threshold is at a low level identically.

Finally, the critical value of a microservice after updating redundancy schema is influenced. Here, we take an example with workflow 8. In Figure 11, none redundancy means all the microservices in the workflow is without redundancy. Redundancy = [3] and Redundancy = [3, 3] mean the third microservice has once and twice the redundancy. Redundancy = [3, 4] means the third and the fourth microservice in the workflow has redundancy.

As shown in Figure 11, with redundancy, the importance of microservices in service composition decreases. On the one hand, the objective of the optimization is to reduce the importance of a microservice. On the other hand, its

objective is to reduce the negative impact of a microservice on microservice composition. From this simulation, we can conclude that the importance of a microservice decreases with the increase of redundancy.

## 7. Conclusion

In the cloud computing environment, the reliability dynamically occurs at runtime, which also has a dynamic impact on the whole application. It leads to the concept drift of probability distribution of reliability time series of a microservice. The evolution of the reliability time series for a microservice follows a continuous time-homogeneous 1st-order Markov Chain evolution rule.

The fluctuation of cloud applications at a time series is uncertain at neighboring time points. So, the effective reliability fluctuation measurement method must evaluate the impact of the uncertainty.

This paper proposes a ranking-based framework to build fault-tolerant cloud applications.

(i) We first model the components in MOCA and cloud application reliability requirement with PrT net. In order to identify the critical microservices, an improved PageRank algorithm has been proposed with request frequency and microservice relation through three steps, i.e., graph building, criticality ranking, and microservice determination

(ii) Then, the multiple temporal transitions of microservice reliability influence the microservice composition in the cloud environment, and the perturbation function describes the single-step transition of microservice reliability in a cloud application. The reliability degradation of a microservice may pose a more significant potential threat to the cloud application. The impact of microservice failures on the MOCA reliability is measured

(iii) Finally, we present a novel FTCARE strategy to determine the most suitable redundancy for a microservice after analyzing the perturbation and degradation properties of a microservice. The microservice reliability properties in microservice composition are iteratively updated by ranking after redundancy. The reliability of the redundancy scheme is the probability that at least one microservice completes successfully

We plan to further expand our work, mainly around the following two directions. Firstly, different fault-tolerant strategies are studied so that each key microservice can be deployed more suitably. Secondly, the microservice based on environmental diversity is combined with the reliability prediction method to improve effectiveness further.

## Data Availability

The data used to support the findings of this study are included within the article.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

## Acknowledgments

This work is partially supported by the NSF of China under no. 61772200.

## References

[1] A. Samanta and J. Tang, "Dyme: dynamic microservice scheduling in edge computing enabled IoT," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6164–6174, 2020.

[2] G. Xie, Y. H. Wei, Y. Le, and R. Li, "Redundancy minimization and cost reduction for workflows with reliability requirements in cloud-based services," *IEEE Transactions on Cloud Computing*, vol. 4, no. 8, pp. 2351–2369, 2020.

[3] Y. Yin, Z. Cao, Y. Xu, H. Gao, R. Li, and Z. Mai, "QoS prediction for service recommendation with features learning in mobile edge computing environment," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 4, pp. 1136–1145, 2020.

[4] Y. Z. Huang, H. H. Xu, H. H. Gao, and W. Hussain, "SSUR: an approach to optimizing virtual machine allocation strategy based on user requirements for cloud data center," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 2, pp. 670–681, 2021.

[5] D. M. Vincenzo and K. Dragi, "Multi-objective scheduling of extreme data scientific workflows in Fog," *Future Generation Computer Systems*, vol. 106, pp. 171–184, 2020.

[6] C. Clab, A. Ms, Z. B. Min, and Y. L. Luo, "Effective replica management for improving reliability and availability in edge-cloud computing environment," *Journal of Parallel and Distributed Computing*, vol. 143, pp. 107–128, 2020.

[7] R. Florin, A. Ghazizadeh, P. Ghazizadeh, S. Olariu, and D. C. Marinescu, "Enhancing reliability and availability through redundancy in vehicular clouds," *IEEE Transactions on Cloud Computing*, vol. 15, no. 4, pp. 2654–2674, 2019.

[8] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-aware online scheduling for real-time workflows in cloud service environment," *IEEE Transactions on Services Computing*, vol. 4, no. 5, pp. 1311–1334, 2019.

[9] N. Kherraf, S. Sharafeddine, C. M. Assi, and A. Ghrayeb, "Latency and reliability-aware workload assignment in IoT networks with mobile edge clouds," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1435–1449, 2019.

[10] H. Gao, W. Huang, and Y. Duan, "The cloud-edge based dynamic reconfiguration to service workflow for mobile ecommerce environments: a QoS prediction perspective," *ACM Transactions on Internet Technology*, vol. 13, no. 4, pp. 3469–3484, 2021.

[11] D. A. Firas and A. M. Mazlina, "Towards agent-based petri net decision making modelling for cloud service composition: a literature survey," *Journal of Network and Computer Applications*, vol. 130, pp. 14–38, 2019.

[12] W. Ha, "Reliability prediction for Web service composition," in *2017 13th International Conference on Computational Intelligence and Security (CIS)*, pp. 570–573, Hong Kong, China, 2017.

[13] G. S. Fan, H. Q. Yu, L. Q. Chen, and D. M. Liu, "Petri net based techniques for constructing reliable service composition," *Journal of Systems and Software*, vol. 86, no. 4, pp. 1089–1106, 2013.

[14] J. Huang, J. Liang, and S. Ali, "A simulation-based optimization approach for reliability-aware service composition in edge computing," *IEEE Access*, vol. 8, pp. 50355–50366, 2020.

[15] Z. G. Zang, Q. Wen, and K. Xu, "A fault tree based microservice reliability evaluation model," *IOP Conference Series: Materials Science and Engineering*, vol. 569, pp. 032–069, 2019.

[16] R. W. Xiao, Z. W. Wu, and D. Y. Wang, "A finite-state-machine model driven service composition architecture for internet of things rapid prototyping," *Future Generation Computer Systems*, vol. 99, pp. 473–488, 2019.

[17] R. M. Pathan and J. Jonsson, "FTGS: fault-tolerant fixed-priority scheduling on multiprocessors," in *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 1164–1175, Changsha, China, 2011.

[18] G. Yao, Y. Ding, and K. Hao, "Using imbalance characteristic for fault-tolerant workflow scheduling in cloud systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3671–3683, 2017.

[19] L. P. Zhao, Y. Ren, and K. Sakurai, "Reliable workflow scheduling with less resource redundancy," *Parallel Computing*, vol. 39, no. 10, pp. 567–585, 2013.

[20] A. R. Setlur, S. J. Nirmala, H. S. Singh, and S. Khoriya, "An efficient fault tolerant workflow scheduling approach using replication heuristics and checkpointing in the cloud," *Journal of Parallel and Distributed Computing*, vol. 136, pp. 14–28, 2020.

[21] N. Kumar, J. Mayank, and A. Mondal, "Reliability aware energy optimized scheduling of non-preemptive periodic real-time tasks on heterogeneous multiprocessor system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 871–885, 2020.

[22] S. Safari, M. Ansari, G. Ershadi, and S. Hessabi, "On the scheduling of energy-aware fault-tolerant mixed-criticality multicore systems with service guarantee exploration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2338–2354, 2019.

[23] H. Gao, C. Liu, Y. Li, and X. Yang, "V2VR: reliable hybrid-network-oriented V2V data transmission and routing considering RSUs and connectivity probability," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 6, pp. 3533–3546, 2021.

[24] A. Sharif, M. Nickray, and A. Shahidinejad, "Energy-efficient fault-tolerant scheduling in a fog-based smart monitoring application," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 36, no. 1, 2020.

[25] J. Yao, Q. Lu, H. Jacobsen, and H. Guan, "Robust multi-resource allocation with demand uncertainties in cloud scheduler," in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pp. 34–43, Hong Kong, China, 2017.

[26] B. Ray, A. Saha, S. Khatua, and S. Roy, "Proactive fault-tolerance technique to enhance reliability of cloud service in cloud federation environment," *IEEE Transactions on Cloud Computing*, vol. 6, no. 8, pp. 2247–2264, 2020.

[27] G. Fan, L. Chen, H. Yu, and D. Liu, "Modeling and analyzing dynamic fault-tolerant strategy for deadline constrained task scheduling in cloud computing," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 4, pp. 1260–1274, 2020.

[28] T. Shi, H. Ma, and G. Chen, "A genetic-based approach to location-aware cloud service brokering in multi-cloud environment," in *2019 IEEE International Conference on Services Computing (SCC)*, pp. 146–153, Milan, Italy, 2019.

[29] L. Wang, Q. He, D. Gao, J. Wan, and Y. Zhang, "Temporal-perturbation aware reliability sensitivity measurement for adaptive cloud service selection," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1254–1279, 2020.

[30] L. Wang, "Architecture-based reliability-sensitive criticality measure for fault-tolerance cloud applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 11, pp. 2408–2421, 2019.

[31] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "FTCloud: a component ranking framework for fault-tolerant cloud applications," in *2010 IEEE 21st International Symposium on Software Reliability Engineering*, pp. 398–407, San Jose, CA, USA, 2010.

[32] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "Component ranking for fault-tolerant cloud applications," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 540–550, 2012.

[33] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, "Ranking significance of software components based on use relations," *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 213–225, 2005.

[34] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-aware and budget-constrained application replication and deployment in multi-cloud environment," in *2020 IEEE International Conference on Web Services (ICWS)*, pp. 110–117, Beijing, China, 2020.

[35] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1954–1969, 2020.

[36] Z. Zheng, Y. Zhang, and M. R. Lyu, "Cloudrank: a QoS-driven component ranking framework for cloud computing," in *2010 29th IEEE Symposium on Reliable Distributed Systems*, pp. 184–193, New Delhi, India, 2010.