计算机科学
COMPUTER SCIENCE

基于深度强化学习的微服务工作流容侵调度算法

李远博, 扈红超, 杨晓晗, 郭威, 刘文彦

引用本文

李远博, 扈红超, 杨晓晗, 郭威, 刘文彦. 基于深度强化学习的微服务工作流容侵调度算法[J]. 计算机科学, 2025, 52(5): 375-383.

LI Yuanbo, HU Hongchao, YANG Xiaohan, GUO Wei, LIU Wenyan. Intrusion Tolerance Scheduling Algorithm for Microservice Workflow Based on Deep Reinforcement Learning [J]. Computer Science, 2025, 52(5): 375-383.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240500033

基于深度强化学习的微服务工作流容侵调度算法

李远博[1,2]　扈红超[1]　杨晓晗[1]　郭　威[1]　刘文彦[1,3]
1 信息工程大学信息技术研究所　郑州 450000
2 洛阳理工学院计算机学院　河南 洛阳 471000
3 网络空间安全教育部重点实验室　郑州 450000
(200900501775@lit.edu.cn)

摘　要　随着微服务和容器技术的快速发展,云中执行的应用可以由多个具有依赖关系的微服务共同完成。然而,基于容器云的微服务由于共享资源而面临许多安全威胁。云中的攻击者可以通过侧通道、容器逃逸方式直接或间接地破坏它们,从而导致产生不正确的输出结果,这将给云中的用户带来巨大的损失。因此,在容器云环境下,提出了一种基于深度强化学习的微服务工作流容侵调度算法(ITSAMW),以提高系统的安全性。首先,该算法为每个微服务构建3个副本,并利用投票裁决机制保证安全性。算法研究了如何调度这些微服务副本,并证明了微服务入侵容忍调度需要满足的位置约束条件。其次,构建了微服务调度和完成时延模型,重新对微服务的安全调度问题进行了形式化描述定义,并利用深度强化学习的方法对问题进行了求解。最后,为了验证算法的有效性,使用Kubernetes搭建了容器云仿真平台,并使用入侵容忍度、完成时延和负载均衡性来对其进行评估。实验结果表明,与现有方法相比,ITSAMW在完成时延增加了17.6%的条件下,入侵容忍度提高了28.1%,负载均衡度降低了13.7%。

关键词:微服务;容器云;工作流;入侵容忍;深度强化学习
中图分类号　TP393.08

Intrusion Tolerance Scheduling Algorithm for Microservice Workflow Based on Deep Reinforcement Learning

LI Yuanbo[1,2],HU Hongchao[1],YANG Xiaohan[1],GUO Wei[1] and LIU Wenyan[1,3]
1 Institute of Information Technology,Information Engineering University,Zhengzhou 450000,China
2 School of Computer Science,Luoyang Institute of Science and Technology,Luoyang,Henan 471000,China
3 Key Laboratory of Cyberspace Security,Ministry of Education,Zhengzhou 450000,China

Abstract　With the rapid development of microservices and container technology,applications executed in the cloud can be completed by multiple microservices with dependencies. However,microservices for container clouds face many security threats due to shared resources. Attackers in the cloud can destroy them directly or indirectly through side channels,container escape,resulting in incorrect output results,which will bring huge losses to users in the cloud. Therefore,an intrusion tolerance scheduling algorithm for microservice workflow(ITSAMW) is proposed to improve the security of the system under the container clouds. Firstly,ITSAMW builds three replicas of each microservice and uses a voting mechanism to guarantee security. ITSAMW studies how to schedule these microservice replicas and proves the location constraints that microservice intrusion tolerance scheduling needs to meet. Secondly,it constructs a microservices scheduling and completion delay model,redefines the security scheduling problem of microservices,and solves the problem with deep reinforcement learning. Finally,in order to verify the effectiveness of ITSAMW,experiments are conducted by using the container clouds simulation platform that Kubernetes builds and are evaluated by using intrusion tolerance,completion delay and load balancing. Experimental results show that compared with the existing methods,under the condition that the completion delay of ITSAMW is increased by 17.6%,the intrusion tolerance is increased by 28.1%,and the load balancing is reduced by 13.7%.

到稿日期:2024-05-09　返修日期:2024-09-14
基金项目:国家自然科学基金(62072467);国家重点研发计划(2021YFB1006201);河南省科技攻关项目(242102210127);河南省重大科技专项(221100211200-02)
This work was supported by the National Natural Science Foundation of China(62072467),National Key Research and Development Program of China(2021YFB1006201),Science and Technology Research Project of Henan Province(242102210127) and Major Science and Technology Special Projects of Henan Province(221100211200-02).
通信作者:扈红超(hhcndsc@163.com)

Keywords Microservices, Container cloud, Workflow, Intrusion tolerance, Deep reinforcement learning

1 引言

随着云计算和容器技术的发展[1]，具有弹性资源供应和快速应用迭代的云应用逐渐发展起来，云中执行的应用被解耦为一系列具有依赖关系的微服务[2-3]。单个微服务通常实现单一功能，众多微服务通过轻量级的通信接口形成微服务工作流，共同提供服务。部署在容器中的微服务具有轻量级、自治和松耦合的属性，它提供了一种独立开发、发布、运维和迭代的敏捷软件开发模式，并鼓励 DevOps 团队的工作[4-5]。与传统的云应用相比，微服务具有性能优势，越来越多的企业系统演变为微服务。尽管有以上这些部署优势，但是微服务工作流存在严重的安全问题[6]。首先，微服务工作流通常被建模为一个有向无环图模型，微服务之间有逻辑依赖关系，任何一个微服务执行错误或受到攻击都会导致错误继承问题，影响最终的应用输出结果。其次，云通常采用多租户共存模式，云服务提供商通常会在同一个计算节点上放置多个容器微服务，它们共享基础设施计算资源。因此，构建和部署微服务系统时，安全性和隐私保护问题非常重要，这也是我们考虑这项工作的重要原因之一。容器技术和微服务架构在改变云端应用设计部署和运行模式的同时，也带来了新的安全威胁。例如：基于容器的虚拟化技术使得同一宿主机上的多个容器共享操作系统内核，给攻击者在集群中进行横向攻击提供了便利；恶意租户可通过微服务的容器的功能或访问控制机制的漏洞（如 CVE-2017-15014，CVE-2017-9150，CVE-2019-5736）进行非法访问，对部署在同一宿主机的其他容器做出提取敏感信息、构建隐蔽信道、干扰运行功能等恶意行为。因此，基于容器运行的应用程序更易遭受攻击者攻击。

传统的网络安全策略主要使用基于边界部署的防护方案，如防火墙、入侵检测等。然而，容器云环境下，传统应用软件的边界逐渐模糊化，防火墙、入侵检测等防护设备的部署位置难以确定。例如：利用 CVE-2019-5736 漏洞可以攻击容器中的 runc 模块，实现容器逃逸。因此，传统基于边界的防护模型无法完全应对容器云环境下的安全威胁。

面对容器云环境下的安全威胁，最直接的策略是通过对软件堆栈或优化硬件来增强容器的隔离性。该技术实现难度较大，且需要对云基础设施进行大范围重建。另一种策略是优化调度机制，其目标是尽可能降低容器攻击、逃逸和同驻带来的安全威胁。基于此，我们提出了 ITSAMW 算法来应对上述安全问题，该安全调度策略是对传统网络安全防护方案的进一步补充。

本文的贡献主要集中在以下几个方面。

(1)分析了微服务工作流存在的安全威胁，并通过实例说明如果采用不适当的调度算法，一个受攻击的计算节点就会破坏微服务工作流的入侵容忍机制。同时，研究了如何调度这些微服务副本，并证明了微服务入侵容忍调度需要满足的位置约束条件。

(2)构建了微服务调度模型、时延模型和负载均衡模型，重新对微服务工作流的安全调度问题进行了形式化描述定义。为解决微服务和节点数量较大时优化复杂度较高的问题，设计了深度强化学习的算法对问题进行了求解。

(3)分别从微服务工作流的入侵容忍度、完成时延和负载均衡性方面来评估调度算法的有效性。实验结果表明，与现有方法相比，ITSAMW 入侵容忍度提高了 28.1%左右，完成时延增加了 17.6%左右，负载均衡度降低了 13.7%左右。

2 相关工作

微服务工作流调度是个 NP 难问题，目前的研究主要集中在满足一定约束条件下的优化工作流任务调度问题。特别地，Wen 等[7]提出了一个新的工作流调度策略，以最小化执行成本，同时解决运行时任务故障问题。Zhou 等[8]提出了一种新的异构最早完成时间算法，该算法利用任务的优先级约束，以最小化工作流的执行成本和完成时间为优化目标。Wu 等[9]提出了一种基于列表调度的启发式算法，该算法在截止日期约束条件下，以最小化执行成本为优化目标。Arabnejad 等[10]开发了一个动态工作流负载调度器，为了处理云中随机到达的多个工作流，以最小化完成时间和执行成本为优化目标。Zhou 等[11]提出了一种云边异构资源配置框架，该框架利用 Lyapunov 优化技术和贪心策略，实现了性能和成本之间的平衡。针对微服务工作流调度，Wang 等[12]提出了两级资源结构调度模型，构建了容器和虚拟机两级资源结构，并提出了融合任务调度和自适应弹性扩展的调度算法。Li 等[13]考虑了虚拟机资源的类型、价格，用线性规划的方法构建了微服务调度数学模型，并提出了弹性工作流调度算法。上述方法都可以有效地降低工作流应用程序的完成时间和执行成本，但没有考虑微服务的安全性问题。

对于微服务的安全性和可靠性，受到免疫系统的启发，Yao 等[14]提出了一种基于免疫系统的工作流调度算法，以避免由于资源故障导致的基于云的工作流中断。为了确保云服务的持续可用性，Gill 等[15]在云资源管理中提出了一种自我保护的方法来防御安全攻击。Yao 等[16]提出了容错工作流调度算法，该算法将工作流级别的总截止日期划分为子任务级别的子截止日期；然后，根据分配的子任务期限，子任务从任务冗余策略和重调度策略中选择合适的容错策略。Zhou 等[17]为了最大限度地降低完成用户请求的货币成本，同时满足对服务质量的要求，利用改进的萤火虫算法优化工作流调度方法，其安全策略是保证中间数据的机密性。Meng 等[18]提出了一种基于安全感知动态调度方法的分布式粒子群优化(PSO)方法用于实时资源分配，目的是在有截止日期和安全约束的情况下最小化成本。

在这些研究中，FTESW[19]和 ITSW[20]与我们的工作有关。FTESW 为服务工作流的每个子任务构建了两个副本，并提出了一种调度这些副本的算法，以提高基于云的服务的可靠性。然而，FTESW 的目的是实现容错，虽然它可以解决系统故障引起的意外故障，但不能解决网络攻击引起的恶意故障。在基于云的服务工作流场景中，网络攻击不仅会导致中断，还会导致服务工作流产生错误的结果。对于这个问题，

ITSW 构建了 3 个副本,并使用投票机制来保证服务工作流的安全性。但是 ITSW 忽略了如何调度这些副本的问题,这个问题影响了基于云的服务工作流的入侵容忍度。如果攻击者成功渗透到执行微服务的虚拟机中,并篡改相关数据以产生错误输出,则上述工作都无法有效解决此问题。因此,我们提出了 ITSAMW 算法来解决这一问题。

3 问题分析

3.1 安全威胁

在云计算环境下,应用层的任务被拆分为多个功能独立、业务耦合的微服务。其中,单个微服务实现单一的业务功能,多个微服务通过开放接口形成服务链,共同完成复杂的应用层任务。

在 ITSW 中,每个子任务 A_1 被复制为 3 个副本 A_1^1(第一个副本)、A_1^2(第二个副本)和 A_1^3(第三个副本),它们将由 3 个异构节点来执行。例如:异构计算节点可以采用不同的操作系统异构生成不同的镜像,在容器运行时层面采用不同的容器运行时实现多个异构节点。同时,每个子任务分别分发 3 个副本到 3 个异构节点中来执行任务。为了在检查中间结果的同时实现不间断微服务工作流的执行,ITSW 采用了滞后决策机制。我们用图 1 来说明其原理。在图 1 中,子任务 A_1 被复制为 3 个副本 A_1^1,A_1^2 和 A_1^3,使用 $N_p(A_1^1)$ 表示分配给 A_1^1 的计算节点,A_1^1 产生的中间数据将立即传输到 A_2^1 中。除了发送中间数据外,还将发送元数据,其中包括哈希值和一个标志,该标志是“立即执行”或“等待决策结果”。在图 1 中,A_1^1 发送给 A_2^1 中的标志为立即执行,A_1^1 发送给 A_2^2 和 A_2^3 中的标志为等待决策结果。假设系统内最多有一个计算节点被攻破,ITSW 产生两个输出,第一个输出依赖于所有子任务的第一个副本,第二个输出依赖于所有子任务的所有副本,通过裁决投票机制将这个结果作为最终的结果。

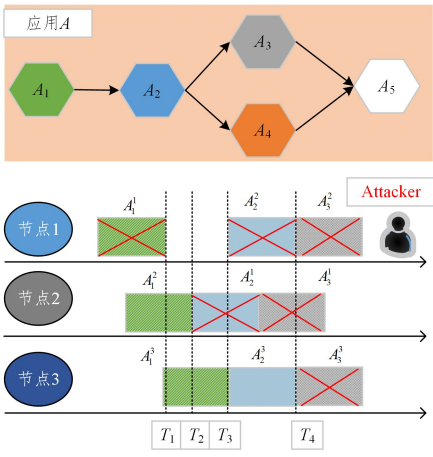


图 1 不恰当的调度实例

Fig. 1 Inappropriate scheduling instance

系统采用多副本异构备份机制提升任务可靠性,在任务调度过程中,系统将每个子任务分配到不同的异构节点执行,同时通过副本的负载均衡调度策略,有效应对节点故障导致的服务中断。通过分布式冗余部署,显著增强了系统容错能力和任务执行稳定性。因此,一个关键节点出现故障,不会影响到整个子任务的执行。同时,定义了中间数据的置信度,其等于执行任务 A_1 得到相同结果的数量与并行执行任务 A_1 的总数量的比值。当置信度小于 1 时,裁决模块中的负反馈机制能够检测出哪个节点出现异常或故障,并通过动态迁移、动态清洗策略来恢复异常节点。

一个调度实例如图 1 所示。假设云中有 3 个可用的容器资源,子任务 A_1^1 被分配到节点 1 中的容器执行,子任务 A_1^2 被分配到节点 2 中的容器执行,子任务 A_1^3 被分配到节点 3 中的容器执行。在 T_1 时刻,子任务 A_1^1 已经完成,生成的中间结果将立即送到节点 2 中来执行。在 T_2 时刻,子任务 A_1^2 已经完成,它需要等待子任务 A_1^3 完成;在 T_3 时刻,子任务 A_1^3 已经完成,对 3 个副本的执行结果进行裁决之后,将最终的执行结果 A_1^{fin} 送到节点 1 和 3 中的容器执行。假设节点 1 的已经被攻击者攻破,由于子任务 A_1^1 在节点 1 中容器执行,因此子任务 A_1^1 的执行结果异常。在 T_3 时刻,子任务 A_1^2 和 A_1^3 的执行结果正确,裁决之后的最终结果是正确的。然而,由于子任务 A_2 执行依赖于 A_1 的执行结果,此时,子任务 A_2^1 被分配到节点 2 的容器中执行,子任务 A_2^2 被分配到节点 1 的容器中,子任务 A_2^3 被分配到节点 3 中容器执行。在 T_4 时刻,由于子任务 A_2^1 依赖于 A_1^1,因此执行结果异常,且节点 1 已经被攻击者捕获,导致子任务 A_2^2 执行结果异常,子任务 A_2^3 执行结果正确。最终,子任务 A_2 的执行结果异常。这是因为 A_2^1 和 A_2^2 产生了一致的错误输出,最终的裁决结果是不正确的。在 T_4 时刻,子任务 A_3 的 3 个副本 A_3^1,A_3^2 和 A_3^3 无论如何进行调度,工作流后面的任务都将是异常的,最终导致应用任务 A 执行异常。

从上述例子可以看出,不恰当的副本调度机制会引起不正确的执行结果,它对系统的入侵容忍度有很大的影响。攻击者控制一个容器,进而通过容器逃逸就可以控制一个计算节点,最终破坏整个应用层程序,系统无法实现入侵容忍。因此,我们提出了一种基于云的微服务工作流容侵调度算法。因为每个微服务的副本分别放到不同的异构计算节点中执行,所以攻击者很难同时入侵多个节点。因此,在设计算法时,仅考虑一个计算节点被攻击者入侵的情况。

3.2 分析微服务的调度位置

本节将介绍在何处调度微服务,以实现系统中的入侵容忍。对于微服务 A_i,很明显,它的 3 个副本 A_i^1,A_i^2 和 A_i^3 中的任何两个都不能分配给同一个计算节点,这个约束可以用式(1)来描述。

$$N_p(A_i^x) \notin \{N_p(A_i^y), y \in \{\{1,2,3\}-x\}\}, x \in \{1,2,3\} \tag{1}$$

其中,$N_p(A_i^x)$ 表示第一个副本 A_i^x 所在的计算节点编号。式(1)仅考虑同一微服务多副本之间的调度约束,但还应考虑具有调用和通信关系的微服务之间的调度约束。

定理 1 微服务 $A_i^x(x \in \{2,3\})$ 不能被调度到前驱微服务的第一个副本节点 $N_p(A_j^1)(A_j \in pred(A_i))$ 中,它可以用式(2)来描述。

$$N_p(A_i^x) \notin \{N_p(A_j^1), A_j \in pred(A_i)\}, x \in \{2,3\} \tag{2}$$

其中,$pred(A_i)$ 表示副本 A_i 的所有前驱微服务集合。

证明:假设 $A_i^x(x \in \{2,3\})$,并且 $A_j^1(A_j \in pred(A_i))$ 被调度放置到同一个计算节点中,如果这个节点被攻击者成功入侵,则微服务 A_j^1 和 $A_i^x(x \in \{2,3\})$ 的结果都是异常的。然而,

A_i^1 的结果取决于 A_j^1 的计算结果，A_j^1 的异常将导致 A_i^1 的结果异常。因此，微服务 A_i 的 3 个副本中有两个是不正确的，这将导致裁决机制失效，输出不正确的结果。

因此，副本的调度放置应该满足式(1)和式(2)。

3.3 威胁模型与假定

本文将应用拆分为微服务，以微服务调用为分析粒度，分析了微服务的调度位置和利用裁决机制保证系统的内生安全性。因此，本文假定的条件如下：

(1)攻击者可以通过微服务工作流的调用关系进行横向攻击；

(2)云服务提供商无法识别租户计划部署的微服务是否存在安全风险；

(3)云应用中拆分为微服务工作流，它们之间的通信相互可信；

(4)副本之间的裁决输出的结果是可信的。

4 问题建模

本章首先构建微服务的调度模型，接着给出完成应用的时延模型和负载均衡模型，最后给出了算法需要解决问题的形式化定义。本文使用的符号定义如表 1 所列。

表 1 常用符号及其含义

Table 1 Common symbols and their meanings

符号	含义
N	计算节点集合
G	微服务调用链的有向无环图集合
A	微服务节点集合
E	微服务之间的调用关系
D	微服务之间传输的数据
X	微服务到计算节点的映射
C	待调度的微服务容器集合
w_i	容器化微服务占用的资源量
W_j	计算节点的资源总量
$EST(A_i,N_j)$	微服务 A_i 被调度到计算节点 N_j 中的最早开始执行时间
$EFT(A_i,N_j)$	微服务 A_i 被调度到计算节点 N_j 中的最早结束时间
$AT(N_j)$	计算节点 N_j 准备就绪的最早时间
ϕ	计算节点的负载均衡性
$N_p(A_i^x)$	微服务 A_i^x 被调度到计算节点的编号

4.1 微服务工作流调度模型

为优化微服务工作流中的调度位置，本节首先建立微服务工作流调度模型 MWSM(Microservice Workflow Scheduling Model)，如式(3)所示。

$$MWSM=\{N,G,X\} \tag{3}$$

其中，$N=\{N_1,N_2,\cdots,N_n\}$ 表示云服务提供商提供的计算节点集合。

服务调用链可以表示为一个有向无环图 $G=(A,E,D)$。其中，$A=\{A_1,A_2,\cdots,A_n\}$ 表示微服务节点集合；$E=\{e_{i,j}\mid A_i,A_j\in A,e_{i,j}\in\{0,1\}\}$ 表示微服务之间的调用关系，即微服务 A_i 执行结束后需要调用微服务 A_j 的接口，并将执行结果传输给微服务 A_j；同时，定义 $D=\{D_{i,j}\mid A_i,A_j\in A\}$ 表示边 $e_{i,j}$ 的权值，代表需要传输的数据量大小。

$X=\{x_{ij}\}_{|C|\times|N|}$ 为 MWSM 的调度策略，其中 $|C|$ 表示等待调度的所有微服务数量，因此调度策略可以表示为 $X=\{x_{ij}\}_{|C|\times|N|}$，其中每个变量的含义如式(4)所示。

$$x_{ij}=\begin{cases}1, & \text{if } C_i \text{ 调度到计算节点 } N_j\\ 0, & \text{其他}\end{cases} \tag{4}$$

在该应用场景下，云平台需要将微服务调度放置到计算节点中，待放置的微服务容器集合可以表示为 $C=\{C_1,C_2,\cdots,C_m\}=\{A_1^1,A_1^2,A_1^3,\cdots,A_n^1,A_n^2,A_n^3\}$，其中集合大小可表示为式(5)。

$$|C|=\sum_G 3\times|A| \tag{5}$$

在微服务放置过程中，云平台需要考虑：(1)每个服务调用链中的微服务需要复制 3 份进行调度放置；(2)每个微服务及其副本只能放置到一个计算节点中；(3)在计算节点中部署微服务不能超过该计算节点的资源。因此，容器调度策略需要满足式(6)和式(7)的约束条件。

$$\sum_{N_j\in N}x_{ij}=1,\forall C_i\in C \tag{6}$$

$$\sum_{c_i\in C}x_{ij}\cdot w_i\leqslant W_j,N_j\in N \tag{7}$$

其中，w_i 表示云平台为容器化微服务 C_i 分配的资源，W_j 为计算节点 N_j 提供的总资源容量。为了简化处理，本文不区分计算、存储等不同类型的资源，将所有资源统一处理来描述所提的策略。

4.2 时延模型

对于微服务工作流，时延表示从第一个微服务 A_1 开始执行到最后一个微服务 A_n 的执行结束的时间。需要指出的是，本文中的时延忽略了用户请求和结果在网路中的传输时延。定义 $EST(A_i,N_j)$ 表示微服务 A_i 被调度到计算节点 N_j 中的最早开始执行时间；$EFT(A_i,N_j)$ 表示微服务 A_i 被调度到计算节点 N_j 中的最早结束时间。因此，对于用户可以直接访问的微服务，最早开始执行时间为 0，即 $EST(A_1,N_j)=0$。为了计算其他微服务的最早开始执行时间，定义 $D_{i,j}$ 为微服务 A_i 调用 A_j 时的数据，$R_{i,j}$ 为数据传输速率，$T_{i,j}$ 表示微服务间相互调用的时延开销。当微服务 A_i 和微服务 A_j 调度到同一个计算节点时，数据可以通过运行环境进行传输，此时调用的时延开销等于 0；而当微服务 A_i 和微服务 A_j 调度到不同的计算节点时，中间数据需要通过网络来进行传输。因此，$T_{i,j}$ 可以通过式(8)来进行求解。

$$T_{i,j}=\begin{cases}\dfrac{D_{i,j}}{R_{i,j}}, & N_p(A_i)\neq N_q(A_j)\\ 0, & N_p(A_i)=N_q(A_j)\end{cases} \tag{8}$$

微服务工作流的完成时间取决于微服务 A_n 的完成时间。对于微服务工作流，只有收到所有的前驱任务的中间数据后，该子任务才能执行。对于微服务 A_i 的第一个副本 A_i^1，其最早开始和完成时间只取决于前驱服务的第一个副本，因此最早开始执行时间 $EST(A_i^1,N_j)$ 和最早完成时间 $EFT(A_i^1,N_j)$ 可以通过式(9)和式(10)来计算。

$$EST(A_i^1,N_j)=\max\{AT(N_j),\max_{A_{i-1}\in pre(A_i)}\{EFT(A_{i-1}^1,N_k)+T_{i-1,i}\}\} \tag{9}$$

$$EFT(A_i^1,N_j)=w_{i,j}+EST(A_i^1,N_j) \tag{10}$$

其中，$AT(N_j)$ 表示微服务 A_i 所在的计算节点 N_j 准备就绪的最早时间，即该容器转入空闲状态的最早时间；$EFT(A_{i-1}^1)$ 表示前驱服务 A_{i-1}^1 的最早完成时间；$EST(A_i^1,N_j)$ 表示微服务 A_i^1 的最早开始时间；$w_{i,j}$ 表示微服务 A_i 在计算节点 N_j 中的执行时间。由于不同的计算节点中配置的资源不同，因此

执行任务的时间也不同,该时间可以在实际环境中获取。

对于微服务 A_i 的第二个和第三个副本 $A_i^x(x\in\{2,3\})$,其最早开始和完成时间的计算式如式(11)和式(12)所示。

$$EST(A_i^x,N_j)=\max\{AT(N_j),\max_{A_{i-1}\in pre(A_i)}\{\max_{y\in\{1,2,3\}}\{EFT(A_{i-1}^y,N_k)\}+T_{i-1,i}\}\},x\in\{2,3\} \quad (11)$$

$$EFT(A_i^x,N_j)=w_{i,j}+EST(A_i^x,N_j),x\in\{2,3\} \quad (12)$$

通过上述公式,微服务工作流从入口微服务开始迭代,可以求出不同调度策略下的完成时间。如果不采用多副本调度部署策略,整个工作流的完成时间可以表示为式(13)。

$$T_1=\max\{EFT(A_n^1)\} \quad (13)$$

采用多副本调度部署策略,微服务工作流的完成时间可以表示为式(14)。

$$T_2=\max\{EFT(A_n^x)\},x\in\{2,3\} \quad (14)$$

其中,A_n 表示服务调用链的"出口"微服务。由于服务工作流可能会存在多个"出口"微服务,因此最终完成时间为最后一个"出口"微服务执行结束的时间。

4.3 负载均衡模型

云计算节点集群负载均衡性是衡量一个集群资源利用情况的重要指标。计算节点负载过重或者过轻,都会降低资源的使用效率。计算节点集群的负载均衡性,可以使用各个计算节点的资源利用率的差异来表征。例如,资源过载的计算节点与资源闲置的计算节点的资源利用率存在显著差异。而差异越大,说明当前计算节点的资源利用率越不平衡。基于此,本节使用各个计算节点的资源利用率的方差来表征计算节点集群的负载均衡性,如式(15)所示。

$$\phi=\frac{1}{|N|}\sum_{N_i\in N}\left(\sum_{C_i\in C}\frac{x_{ij}\cdot w_i}{W_j}-\frac{1}{|N|}\sum_{N_i\in N}\sum_{C_i\in C}\frac{x_{ij}\cdot w_i}{W_j}\right)^2 \quad (15)$$

4.4 问题定义

本文的研究目标是通过优化微服务及其副本的调度位置 X,来增强微服务的入侵容忍能力,降低攻击者对计算节点造成的危害。根据前面的描述,本文对微服务调度策略的优化目标为保证系统的安全性能,同时降低微服务工作流的时延并优化计算节点的负载均衡性,该问题可表达为式(16)。

$$\begin{aligned}&X^*=\arg\min_X\{\alpha\cdot\delta(T)+(1-\alpha)\cdot\delta(\phi)\}\\&\sum_{N_j\in N}x_{ij}=1,\forall C_i\in C\\&\sum_{C_i\in C}x_{ij}\cdot w_i\leqslant W_j,N_j\in N\\&N_p(A_i^x)\notin\{N_p(A_i^y),y\in\{\{1,2,3\}-x\}\},x\in\{1,2,3\}\\&N_p(A_i^x)\notin\{N_p(A_j^1),A_j\in pred(A_i)\},x\in\{2,3\}\end{aligned} \quad (16)$$

其中,α 为时延与负载均衡之间的相对重要因子,$\delta(\cdot)$ 为两个目标的归一化函数。该问题为 NP 难问题,为了在实际云环境中快速求解微服务的调度策略,本文提出了一种基于强化学习的策略优化方法,通过对模型的充分训练,达到快速生成微服务调度策略的目的。

5 基于 DQN 的 ITSAMW 算法

为解决微服务和计算节点中容器数量较大时优化搜索复杂度较高的问题,本文基于深度学习与强化学习相结合的深度 Q 网络(Deep Q Network,DQN)[21]求解式(16)所示问题,提出了一种基于 DQN 的微服务调度算法。DQN 通过 Q 网络对状态下的动作值进行估计,避免了传统 Q 学习算法在高维状态动作空间中造成的 Q 表爆炸问题。

在 DQN 中,存在两个相互独立的神经网络:评估网络和目标网络,其参数分别用 θ 与 θ^- 表示。在 t 时刻,智能体在状态 s_t 下执行了动作 a_t,观察到收益为 r_t,且下一个状态为 s_{t+1}。四元组 $\langle s_t,a_t,r_t,s_{t+1}\rangle$ 便是 DQN 从环境中学习到的经验。评估网络的参数会基于学习到的经验不断迭代更新,在第 t 时刻,其损失函数可以式(17)来表示。

$$Loss(\theta)=E\left[\frac{1}{2}(y_t-Q(s_t,a;\theta))^2\right] \quad (17)$$

其中,$Q(s_t,a;\theta)$ 表示神经网络参数为 θ 时,输入 s_t 下动作 a 对应的输出值;y_t 是评估网络的学习目标,它由当前时刻的收益值 r_t 和目标网络估计下一时刻的价值 $Q(s_{t+1},a;\theta^-)$ 组成,具体可用式(18)来表示。

$$y_t=r_t+\gamma\max_{a'}Q(s_{t+1},a';\theta^-) \quad (18)$$

在算法中,使用微服务的调度位置作为神经网络的输入,离散化后的策略参数作为神经网络的输出[22]。在算法执行中,迭代次数 STEPS 的选取应该保证神经网络能够收敛。算法中主要涉及 DQN 与环境交互的接口,状态、动作和收益值的具体设计如下。

(1)状态 S_t:微服务调度策略矩阵反映了当前微服务的调度位置,因此状态 S_t 可以表示为 $S_t=X$。

(2)动作 A_t:为了避免 DQN 的动作空间过大而收敛速度慢的问题,将高纬度单步决策分解为多次低纬度决策。低纬度的决策包括两个步骤:第一,选择一个微服务副本;第二,将选中的微服务放置在计算节点 N_j。因此,动作空间可以表示为 $AC=\{i+,i-,j+,j-,0\}$,$A_t\in AC$,其中 i 表示微服务的索引号增加 1 或减少 1,j 表示微服务调度位置的索引号增加 1 或减少 1,0 表示不采取任务动作。

(3)奖励 R_t:奖励值由式(16)所示的目标函数来决定。

基于 DQN 的 ITSAMW 算法的求解过程如算法 1 所示。

算法 1 基于 DQN 的 ITSAMW 算法

输入:MWSM

输出:DQN 神经网络参数

1. 初始化:神经网络的参数 θ 和 θ^-,经验复用池 D,网络更新步长 L,贪婪系数 ε
2. for episode in STEPS:
3. $s_t=X_0$
4. $\varepsilon=\varepsilon-(\varepsilon_{max}-\varepsilon_{min})/M$
5. if$\varepsilon<\varepsilon_{min}$do
6. $\varepsilon=\varepsilon_{min}$
7. end if
8. 以 ε 的概率随机选择一个动作 a_t,否则选择 $a_t=Q(s_{t+1},a';\theta^-)$
9. 基于动作 a_t 修改策略矩阵,到达下一个状态 s_{t+1}
10. 基于式(16)计算奖励值 r_t
11. 将样本经验$\langle s_t,a_t,r_t,s_{t+1}\rangle$存储于经验复用池 D 中
12. if $|D|>$batch do
13. 从 D 中采集 batch 个样本,基于式(17)计算损失函数,使用梯度下降方法更新网络参数 θ
14. end if
15. 使用式(17)和式(18)执行梯度下降,更新网络参数 θ
16. if $|D|\%L==0$ do
17. $\theta^-\leftarrow\theta$
18. end if
19. end for

在训练过程中的样本探索阶段，DQN 根据网络输出的 Q 值估值，基于“ε-贪婪策略”选择将执行的动作，即以 $1-\varepsilon$ 的概率选择当前状态下 Q 值的估值最大的动作，以 ε 的概率随机选择动作。随着训练的进行，ε 从 ε_{max} 开始，分 M 步线性下降至 ε_{min}，使得模型在训练前期更加广泛地探索未尝试过的动作，训练后期更加充分地利用训练所得经验。

为了缩短 DQN 模型的训练时间，提高模型的准确度，针对微服务场景的特点提出了两阶段训练策略。首先进行离线训练，随机生成容器云中计算节点的数量，生成微服务的数量以及微服务之间的调用关系，并基于此组成训练的样本，进行离线训练，直到模型收敛。其次进行在线训练，当微服务运行状态发生变化时，利用离线状态生成的模型参数进行在线优化训练，从而进一步求出最优的安全调度策略。由于采用了两阶段的训练策略，算法能够快速收敛，同时在微服务工作流中，服务的执行时间是时延的主要因素，因此本文忽略了微服务之间的通信时延和 DQN 的训练开销。

6 实验评估

6.1 实验设置

实验平台为 Intel Xeon(R) CPU E5-2680 V3 @ 2.50GHz，48 核，RAM 配置为 32GB 的 DDR 的 NSFocus MICA-9700 服务器。ITSAMW 算法是在 Pytorch 1.12.1 上实现的，语言版本为 Python 3.9。由于本文研究的是容器云环境下微服务工作流的容侵调度问题，因此仿真实验围绕这一目标，重点模拟微服务及其副本的容器调度，而简化一些次要参数的设置。微服务工作流仿真的数据集使用 Networkx 2.8.4 工具包随机生成，微服务数量为 20，计算节点数量为 20，微服务连边概率为 0.4，微服务及其副本占用容器资源量 w_i 均为 1，每个计算节点的资源总量 W_j 均为 100。在具体的实验过程中，可以根据所讨论的问题调整数据集大小。

为了合理地设置 DQN 的模型参数，对微服务工作流的调度模型进行关键参数分析，合理设置模型参数，使得 ITSAMW 算法能够快速收敛。其中，收益折扣因子 γ 为 0.9，批量采样 batch 为 32，网络更新步长 200，经验复用池 D 为 30000，ε 下降所需步数 M 为 10000。对于每个训练步长，DQN 会与环境进行 100 次交互。

图 2 中所示的是经验复用池的大小对 ITSAMW 算法收敛的影响情况。

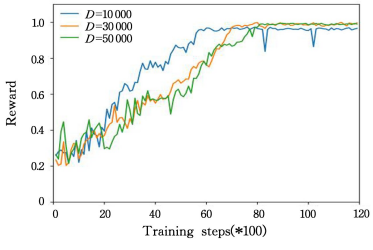


图 2 经验复用池大小对收益值的影响

Fig. 2 Impact of experience replay pool on reward

可以看到，ITSAMW 算法均能在 80 步长收敛。经验复用池 D 的大小影响 Q 网络的稳定性，过小的经验复用池在探索阶段存储的样本较少，当 D 较小时，经验复用池容量不足，可能会丢失某些重要的经验，造成收敛不稳定。例如：当 $D=10000$ 时，训练至 83 和 104 步左右时，模型发生了波动。而 $D=30000$ 与 $D=50000$ 条件下的累计收益值趋于稳定，且二者没有明显区别，因此本文设置 $D=30000$。

图 3 所示的是 ε 从 ε_{max} 下降至 ε_{min} 需要的步数 M 对算法收敛情况的影响。可以看到，随着 ε 的下降，所需步数减小，算法的收敛速度变快，但是 $M=5000$ 时的收益值低于 $M=10000$ 和 $M=20000$ 时的收益。其原因在于 M 反映的是训练过程中由“探索”到“利用”的转变速度，该转变速度越快，则算法对已学习到的动作价值的利用程度越高。但由于训练过程对未知动作的“探索”不足，算法易收敛于次优解。因此，基于实验结果，本文设置 $M=10000$，其收敛速度相对较快，且收益值与 $M=20000$ 的设置无明显差别。

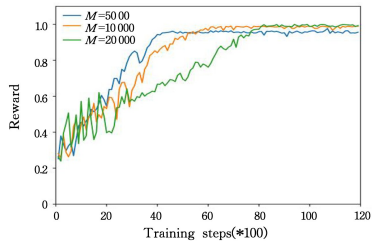


图 3 ε 下降步数对收益值的影响

Fig. 3 Impact of decreasing steps ε on reward

6.2 对比策略

使用以下算法与本文提出的算法进行比较。

(1)FTESW 算法。该算法为每个微服务构建了两个副本：一个主副本，一个备份副本。算法的主要思路是将主副本和备份副本优先调度到不同的计算节点中，后继微服务采用随机调度的方法将其调度到不同的计算节点中，从而提高云服务的可靠性。但是，它没有考虑安全性。

(2)ITSW 算法。该算法为每个微服务构建 3 个副本。算法的主要思路是将相同微服务的 3 个副本优先调度到不同的计算节点中，将一个计算节点充分利用之后，再考虑调度到其他节点。同时，副本之间使用裁决投票机制来提高安全性。但是，它忽略了调度这些副本之间的约束关系。

(3)ITSAMW 算法。该算法为每个微服务构建 3 个副本。算法的主要思路是将相同微服务的 3 个副本优先调度到不同的计算节点中，后继微服务副本之间使用裁决投票机制来提高安全性。同时，调度位置满足约束条件，进一步提升系统的入侵容忍能力。

6.3 实验结果

为了分析 ITSAMW 算法在不同调度情况下的有效性，仿真实验首先利用计算节点数量和每个云应用中的微服务数量这两个参数对不同调度情况进行了仿真，并通过与相关算法的对比来探讨 ITSAMW 算法的有效性。同时，采用 3 个指标来衡量调度算法的安全和执行性能，它们分别是：(1)微服务工作流执行成功率 ESR，它反映了系统的入侵容忍能力；(2)任务的时延开销 T，它反映了系统完成云应用需要的时间开销；(3)负载均衡能力 ϕ，它反映了计算节点的资源利用率。

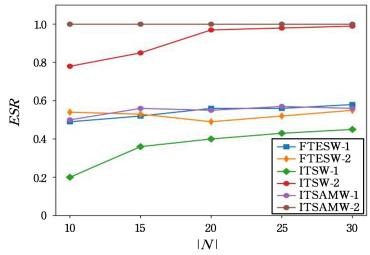
(1)入侵容忍度分析

在本实验中，我们用 ESR 来量化入侵容忍度。在被入侵

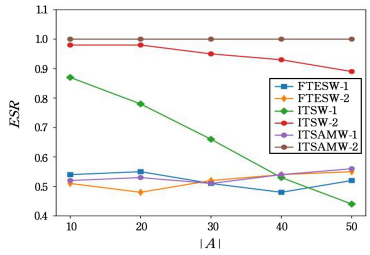
计算节点数量相同的情况下，算法的 ESR 越高，入侵容忍度越强。ESR 与计算节点数量和微服务数量的关系如图 4 所示。对于 FTESW 算法，FTESW-1 表示主副本结果，FTESW-2 表示备份副本结果。对于 ITSW 和 ITSAMW 算法，ITSW-1 和 ITSAMW-1 表示第一个副本结果，ITSW-2 和 ITSAMW-2 表示 3 个副本裁决之后的结果。

对于 FTESW 算法，一部分可用计算节点用于执行主副本，另一部分用于执行备份副本。对于 ITSAMW 算法，一部分可用计算节点用于执行第一个微服务副本，另一部分用于执行微服务第二和第三个副本。对于 ITSW，调度位置没有限制，所有可用的节点资源都可以用于执行任何微服务副本。对于 ITSAMW，副本之间的调度位置必须满足前面定义的约束条件。

假设有一个计算节点被攻击者捕获，任何分配给此节点的微服务都将输出不正确的结果。在这种情况下，我们对不同可用计算节点下的 ITSAMW，FTESW 和 ITSW 的平均 ESR 进行了评价，结果如图 4 所示。如果受攻击的计算节点属于执行主子任务副本的集群，则 FTESW-1 比 FTESW-2 出错的概率更高。同样，如果受攻击的计算节点属于执行备份子任务副本的集群，则 FTESW-2 错误的概率比 FTESW-1 更高。



(a)计算节点数量对 ESR 的影响



(b)微服务数量对 ESR 的影响

图 4 计算节点数量和微服务数量对 ESR 的影响

Fig. 4 Impact of the number of computing nodes and microservices on ESR

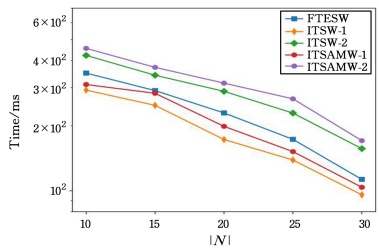
分析图 4(a)可知，保持微服务数量为 20 时，ITSW-2 的 ESR 随着计算节点数量的增加逐渐增大，它的 ESR 并不总是等于 1.0，这验证了第 3.1 节中指出的调度问题，即不恰当的微服务调度位置会影响系统的入侵容忍度。同时，由于节点数量的增加，微服务调度到被攻破的计算节点的概率降低，系统的整体入侵容忍度呈上升趋势。ITSAMW 修复了 ITSW 的弱点，可以容忍一个被攻破的计算节点，即使将微服务调度到被攻击的计算节点，调度位置的约束和裁决机制也能保证最终输出的正确性，所以 ITSAMW-2 的 ESR 总是等于 1.0。

分析图 4(b)可知，保持计算节点数量为 20 时，随着微服务工作流中微服务数量的增加，FTESW 算法的 ESR 没有太大变化，这是因为该算法取决于副本分配到受攻击的计算节点属于执行主副本或者备用副本的集群。ITSW-1 的 ESR 随着微服务数的增加持续下降，这是因为微服务数量较少时，第一个副本分配到受攻击节点的概率较小，随着微服务数量的增加，其分配到受攻击节点的概率持续增加，因此 ITSW-1 的 ESR 呈下降趋势。ITSW-2 的 ESR 下降幅度比较小，主要是因为该算法采用了裁决机制，保证了最终结果的正确性，但它的 ESR 不总是等于 0。与 FTESW 和 ITSW 相比，ITSAMW 的 ESR 平均提高了 28.1%。

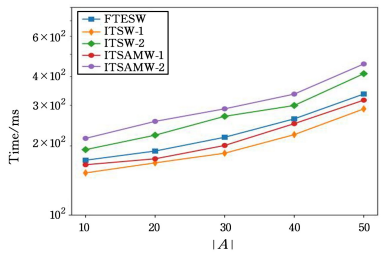
(2)完成时延分析

图 5(a)中显示了保持微服务数量为 20 时，微服务工作流的完成时延随着计算节点的变化情况。FTESW-1 和 FTESW-2 由于完成时间基本接近，不再加以区分，用 FTESW 统一表示。在相同的计算节点下，FTESW 算法的时间开销最低，因为只要有一个副本完成任务，就完成了整个工作流的执行。ITSAMW-1 的时间开销介于 FTESW 算法和 ISTW-1 之间，因为 ISTW 采用了优先调度的策略，使一个计算节点尽可能地充分利用之后，再考虑调度放置到其他节点，目的是增加节点的使用效率，减少节点的租用成本。ITSAMW 算法在进行强化学习时，优化目标中同时加入了负载均衡性考虑，尽可能地将任务调度到不同的计算节点中去，增加了相邻微服务之间的传输时延，因此总体的完成时延增加。ITSAMW-2 和 ISTW-2 的时间开销相比 ITSAMW-1 和 ISTW-1 有所增加，因为它们都需要等待 3 个副本的执行裁决结果，这也是为保证安全性、提升系统入侵容忍能力付出的代价。

分析图 5(b)可知，保持计算节点数量为 20 时，随着微服务数量的增加，时间开销整体呈现上升趋势。ITSAMW 算法的时间开销介于 FTESW 和 ITSW 之间，相对于 ITSW 算法，ITSAMW 时延开销平均增加了 17.6%，但它比 FTESW 算法和 ITSW 算法具有更好的入侵容忍能力。



(a) 计算节点数量对时延影响



(b) 微服务数量对时延影响

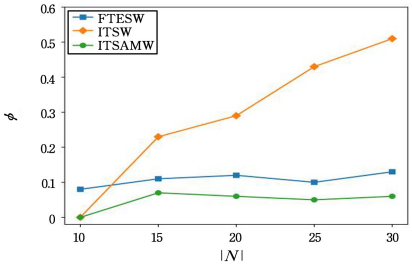
图 5 计算节点数量和微服务数量对时延的影响

Fig. 5 Impact of the number of computing nodes and microservices on time

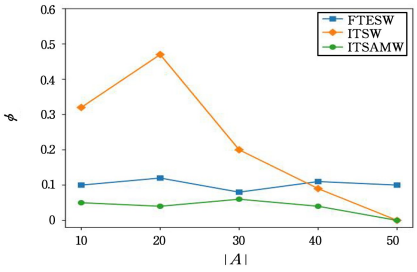
(3)负载均衡性分析

分析图 6(a)可知,保持微服务的数量为 20 时,随着计算节点的增加,ITSW 算法的负载均衡度整体呈现出上升的趋势。其原因在于式(15)将计算节点的负载均衡度定义为所有计算节点的资源利用率的方差。差异越大,说明当前微服务调度策略导致计算节点的资源利用率越不均衡。ITSW 算法是将微服务的副本优先调度到已经分配任务的计算节点中,将一个计算节点充分利用之后再考虑调度到其他计算节点。随着计算节点数量的增加,ITSW 算法调度微服务的策略引起资源利用率为 0 的计算节点越来越多,各个计算节点的资源利用率之间的差异性升高,与式(15)显示的一致。FTESW 算法和 ITSAMW 算法的负载均衡度低于 ITSW 算法,其主要原因在于 FTESW 算法对微服务副本的调度采用了随机放置策略,该策略基于生成的随机数值将微服务调度到计算节点中,使得计算节点资源利用率相对更加均衡。ITSAMW 算法采用强化学习方法优化微服务调度策略,其奖励函数中考虑了计算节点的资源利用率,因此其负载均衡度也相对更好。

分析图 6(b)可知,保持计算节点的数量为 20 时,随着微服务数量的增加,ITSW 算法的负载均衡度呈现出先升高后下降的趋势。其原因在于当 ITSW 算法需要调度的微服务数量较少时,调度微服务的计算节点少于空闲的计算节点。随着计算节点调度的微服务越来越多,节点之间的资源利用率的差异越来越大,负载不均衡度逐渐升高。随着微服务数量的持续增加,空闲节点的资源利用率逐渐增大,与其他满载之间的计算节点的利用率差异逐渐减小,因此其负载不均衡度降低。相对于 FTESW 和 ITSW 算法,ITSAMW 的负载均衡度平均降低了 13.7%。



(a) 计算节点数量对负载均衡性的影响



(b) 计算节点数量和微服务数量对负载均衡性的影响

图 6 微服务数量对负载均衡性的影响

Fig. 6 Impact of the number of computing nodes and microservices on ϕ

结束语 基于云的微服务工作流调度面临着许多安全威胁,然而,目前微服务调度的研究大多集中在资源失效引起的故障问题,而不是网络攻击。与资源故障相比,网络攻击的危害更大。ITSW 为基于云的工作流调度设计了一个入侵容忍框架,但该框架没有考虑微服务副本之间的调度位置问题,削弱了系统的安全性。因此,在 ITSW 框架的基础上,我们提出了 ITSAMW。首先,用一个例子来分析了 ITSW 的安全威胁。然后,根据微服务副本之间的依赖关系分析了调度位置的约束条件,以提高系统的入侵容忍度。在 ITSAMW 中使用这些约束来确定微服务调度到合适的计算节点,同时考虑应用完成时间和计算节点的负载均衡性。此外,我们还提出了一种基于 DQN 的微服务工作流容侵调度算法,其可以根据微服务状态的变化,在保证安全性的同时,动态自适应地获取最优的调度策略,降低微服务的完成时延,优化计算节点的负载均衡性。最后,通过仿真实验,评估了算法的性能。实验结果表明,与 ITSW 相比,ITSAMW 不仅能有效提升基于云的微服务调度的入侵容忍度,而且能优化完成时间和节点的负载均衡性。为了进一步推进工作,未来将在真实的云环境下构建 ITSAMW 的原型系统,并测试其安全性;研究算法能够容忍更多的受攻击的计算节点,并探索节点数量和微服务调度算法之间的关系。

参 考 文 献

[1] ZHOU X, PENG X, XIE T, et al. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study[J]. IEEE Transactions on Software Engineering, 2021, 47(2): 243-260.

[2] KHAN M, TAHERI J, Al-DULAIMY A, et al. PerfSim: A Performance Simulator for Cloud Native Computing[J]. IEEE Transactions on Cloud Computing, 2021, 11(2): 1395-1413.

[3] AROUK O, NIKAEIN N. Kube5G: A Cloud-Native 5G Service Platform[C]// Proceedings of Global Communications Conference(GLOBECOM). IEEE, 2020: 1-8.

[4] ZHAO P, WU L, HONG Z, et al. Research on Multi-cloud Access Control Policy Integration Framework[J]. China Communications, 2019, 16(9): 222-234.

[5] PEREIRA-VALE A, FERNANDEZ E B, MONGE R, et al. Security in Microservice-based Systems: A Multivocal Literature Review[J]. Computers & Security, 2021, 103: 102200.

[6] LI C, LIU J, WANG M, et al. Fault-tolerant Scheduling and Data Placement for Scientific Workflow Processing in Geo-distributed Clouds[J]. Journal of Systems and Software, 2022, 187: 111227.

[7] WEN Z, QASHA R, LI Z, et al. Dynamically Partitioning Workflow Over Federated Clouds for Optimising the Monetary Cost and Handling Run-time Failures[J]. IEEE Transactions on Cloud Computing, 2020, 8(4): 1093-1107.

[8] ZHOU X, ZHANG G, SUN J, et al. Minimizing Cost and Makespan for Workflow Scheduling in Cloud Using Fuzzy Dominance Sort Based HEFT[J]. Future Generation Computer Systems, 2019, 93: 278-289.

[9] WU Q, ISHIKAWA F, ZHU Q, et al. Deadline-Constrained Cost Optimization Approaches for Workflow Scheduling in Clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(12): 3401-3412.

[10] ARABNEJAD V,BUBENDORFER K,NG B. Dynamic Multi-workflow Scheduling:A Deadline and Cost-aware Approach for Commercial Clouds[J]. Future Generation Computer Systems,2019,100:98-108.

[11] ZHOU Z,YU S,CHEN W,et al. CE-IoT:Cost-effective Cloud-edge Resource Provisioning for Heterogeneous IoT Applications [J]. IEEE Internet of Things Journal,2020,7(9):8600-8614.

[12] WANG S,DING Z,JIANG C. Elastic Scheduling for Microservice Applications in Clouds[J]. IEEE Transactions on Parallel and Distributed Systems,2021,32(1):98-115.

[13] LI W,LI X,RUIZ R. Scheduling Microservice-based Workflows to Containers in on-demand Cloud Resources[C]// 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design(CSCWD). IEEE,2021:61-66.

[14] YAO G,DING Y,REN L,et al. An Immune System-inspired Rescheduling Algorithm for Workflow in Cloud Systems[J]. Knowledge-Based Systems,2016,99:39-50.

[15] GILL S S,BUYYA R. SECURE:Self-protection Approach in Cloud Resource Management[J]. IEEE Cloud Computing,2018,5(1):60-72.

[16] YAO G,DING Y,HAO K. Using Imbalance Characteristic for Fault-tolerant Workflow Scheduling in Cloud Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2017,28(12):3671-3683.

[17] ZHOU C,WANG T,LI L,et al. Makespan and Security-aware Workflow Scheduling for Cloud Service Cost Minimization Using Firefly Optimizer[C]// International Conference on Algorithms and Architectures for Parallel Processing. Springer Nature Switzerland,2023:620-641.

[18] MENG S,HUANG W,YIN X,et al. Security-aware Dynamic Scheduling for Real-time Optimization in Cloud-based Industrial Applications[J]. IEEE Transactions on Industrial Informatics, 2021,17(6):4219-4228.

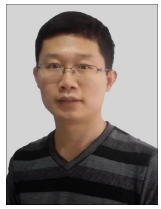
[19] DING Y,YAO G,HAO K. Fault-tolerant Elastic Scheduling Algorithm for Workflow in Cloud Systems [J]. Information Sciences,2018,393:47-65.

[20] WANG Y,GUO Y,GUO Z,et al. Protecting Scientific Workflows in Clouds with an Intrusion Tolerant System[J]. IET Information Security,2020,14(2):157-165.

[21] LI H,GUO Y,SUN P,et al. An Optimal Defensive Deception Framework for the Container-based Cloud with Deep Reinforcement Learning[J]. IET Information Security,2022,16(3):178-192.

[22] ZHOU D,CHEN H,CHENG G. A Security Containers Placement Algorithm Based on DQN for Microservices to Defend Against Co-Resident Threat[C]// 2023 8th International Conference on Computer and Communication Systems (ICCCS). IEEE,2023:683-688.

LI Yuanbo, born in 1988, doctoral candidate. His main research interests include cloud computing, endogenous security and active defense.



HU Hongchao, born in 1982, professor, Ph.D supervisor. His main research interests include cloud computing and network security.

(责任编辑:柯颖)