

## Scientific workflow execution system based on mimic defense in the cloud environment\*

Ya-wen WANG, Jiang-xing WU<sup>†‡</sup>, Yun-fei GUO, Hong-chao HU, Wen-yan LIU, Guo-zhen CHENG

National Digital Switching System Engineering Technology Research Center, Zhengzhou 450002, China

<sup>†</sup>E-mail: JiangXing\_WU\_NDSC@163.com

Received Oct. 7, 2018; Revision accepted Nov. 17, 2018; Crosschecked Dec. 17, 2018

**Abstract:** With more large-scale scientific computing tasks being delivered to cloud computing platforms, cloud workflow systems are designed for managing and arranging these complicated tasks. However, multi-tenant coexistence service mode of cloud computing brings serious security risks, which will threaten the normal execution of cloud workflows. To strengthen the security of cloud workflows, a mimic cloud computing task execution system for scientific workflows is proposed. The idea of mimic defense contains mainly three aspects: heterogeneity, redundancy, and dynamics. For heterogeneity, the diversities of physical servers, hypervisors, and operating systems are integrated to build a robust system framework. For redundancy, each sub-task of the workflow will be executed simultaneously by multiple executors. Considering efficiency and security, a delayed decision mechanism is proposed to check the results of task execution. For dynamics, a dynamic task scheduling mechanism is devised for switching workflow execution environment and shortening the life cycle of executors, which can confuse the adversaries and purify task executors. Experimental results show that the proposed system can effectively strengthen the security of cloud workflow execution.

**Key words:** Scientific workflow; Mimic defense; Cloud security; Intrusion tolerance

<https://doi.org/10.1631/FITEE.1800621>

**CLC number:** TN915.08

### 1 Introduction

In many scientific research fields, such as quantum physics, astronomy, and bioinformatics, the process of scientific computing often consists of tens of thousands of steps, requiring large-scale data analysis and processing (Lv et al., 2015). To properly manage, arrange, execute, and track these steps, the concept of scientific workflows has been presented. As the complexity of the problems to be solved increases, the present large-scale scientific workflows


often need to be implemented on complex distributed computing environments, such as supercomputers, distributed cluster systems, and grid systems. However, constructing such a system is very expensive (Deldari et al., 2017).

With the development of virtualization technologies, the emergence of cloud computing provides a new deployment and implementation solution for scientific workflow applications. The cloud computing, with the resource renting, application deployment, and service outsourcing as the core, builds the computing environments through the integration of distributed resources to meet a variety of service requirements (Juve and Deelman, 2011).

As cloud customers, they no longer need to purchase hardware, and they just need to pay a certain fee to easily access the required computing and storage resources over the Internet. Due to the low cost and the convenient resource access, more and more

<sup>‡</sup> Corresponding author

\* Project supported by the National Natural Science Foundation of China (Nos. 61521003 and 61602509), the National Key Technologies R&D Program of China (Nos. 2016YFB0800100 and 2016YFB0800101), and the Key Technologies R&D Program of Henan Province, China (No. 172102210615)

 ORCID: Ya-wen WANG, <http://orcid.org/0000-0003-4783-0450>  
© Zhejiang University and Springer-Verlag GmbH Germany, part of Springer Nature 2018

large-scale scientific workflows have been delivered to cloud systems for completion (Wang et al., 2014).

However, the cloud systems employ multi-tenant coexistence service mode, which brings both benefits and risks; for example, an attacker can legally rent a virtual machine (VM) and use it as a springboard to steal or tamper with other tenants' data or information (Ainapure et al., 2018). Compared with normal cloud computing tasks, cloud workflow tasks have three salient features: (1) The execution duration is long. A practical cloud workflow task often requires several weeks or even months to be finished (Yuan et al., 2012), which provides sufficient preparation time for attackers. (2) Cloud workflows are computationally intensive tasks (Gupta et al., 2016), which require a high computing accuracy. Any intermediate error will directly lead to the failure of the entire cloud workflow execution. (3) Cloud workflows are often applied in important scientific research fields. Once it happens that the execution results are tampered with without being noticed, it will bring incalculable losses.

Inspired by mimic defense (Hu et al., 2017), the ideas of heterogeneity, redundancy, and dynamics are integrated to construct a mimic cloud workflow execution system, which can significantly reduce the influence of system uncertainty disturbance on the accuracy of calculation results and effectively prevent network attacks that maliciously disrupt workflow execution. The contributions of this study are summarized as follows:

1. We devise the framework of a mimic cloud workflow execution system based on the diversities of operating systems (OSs), hypervisors, and physical servers, which can reduce the common attack surface among the components of the system, preventing fault propagation.

2. We use a heterogeneous task executor cluster to process workflows in parallel and design a decision module to verify the execution results of the executor cluster, achieving intrusion tolerant workflow execution.

3. We present the elastic executor generation and recycling mechanism, shortening the life cycle to keep the pure state of executors.

4. We devise a dynamic workflow execution environment switching strategy, which enables different sub-tasks to be executed in different environments to confuse adversaries.

## 2 Related work

Workflow technology originated from the early 1980s. With the development of distributed computing, business workflows, grid workflows, and cloud workflows have emerged. Cloud computing systems employ the virtualization technology to achieve flexible resource management. Compared with other workflow systems, cloud workflows are more efficient and flexible. Therefore, cloud workflow technologies have become a research hotspot in recent years. Pandey et al. (2010) quantified the overall task scheduling cost by employing task execution costs and access costs between different nodes, and then introduced the particle swarm optimization algorithm to acquire the optimal scheduling strategy, minimizing the overall task execution cost. Lee et al. (2015) proposed an intelligent task scheduling algorithm to improve resource utilization. Casas et al. (2017) presented the balanced and file reuse-replication scheduling (BaRRS) algorithm, which adopts the data replication method to reduce the amount of data transmission at the stage of task scheduling, achieving the improvement of task scheduling efficiency. To reasonably manage the data transmission during task scheduling, Aktas et al. (2014) used software-defined networking (SDN) technologies to manage workflows and resource allocation.

However, most research on cloud workflow has focused on improving the task execution efficiency and optimizing the resource allocation. In recent years, cloud security issues have gradually attracted widespread attention. Some scholars have begun to study cloud workflow security issues. Yao et al. (2016) considered that the process of rescheduling in cloud systems had great similarities to the immune system which kept the body stable by removing the intrusive antigens, so an immune system inspired failure-aware rescheduling algorithm for the workflow task in cloud systems was designed to achieve fault-tolerant workflow execution. Ding et al. (2017) proposed a primary-backup workflow scheduling strategy to realize fault-tolerant elastic task scheduling. Resubmission and replication are two fundamental techniques in distributed computing systems for fault tolerance. Yao et al. (2017) took these two techniques together for fault-tolerant workflow scheduling in cloud systems. However, all the above research used

passive defense methods, which can prevent system failures but cannot effectively resist malicious attacks.

As a kind of active network defense technology, mimic defense provides a new idea for solving cloud workflow security problems. The idea of mimic defense contains mainly three aspects: heterogeneity, redundancy, and dynamics. Many researchers have employed one of them to solve the network security problems.

For heterogeneity, Evans and Thompson et al. (2016) devised a defense mechanism based on OS diversity to strengthen the system security; however, the heterogeneity difference between different OSs was not considered. For this problem, Garcia et al. (2011) used common vulnerabilities to quantify the heterogeneity between different OSs, but they did not mention how to select OSs. Afterwards, Garcia et al. (2014) presented a strategy for OS selection, in which the OS set with the least common vulnerabilities would be selected.

For redundancy, the Byzantine fault tolerance (BFT) method is one of representative methods. On the basis of the BFT method, Platania et al. (2014, 2016) devised a practical intrusion-tolerant replication system. Though the BFT method can improve the system security, multiple consultation steps contained in the BFT method will produce a huge time overhead. To limit the costs of redundant methods, Zheng et al. (2012) proposed a component ranking algorithm, in which redundant methods were used for only the protection of the selected important components. In large-scale SDN, multiple controllers can be deployed in the control plane to strengthen the system reliability. However, it usually involves big data in SDN. At the same time, the legality of the data sources should be ensured. To address these challenges, Wu et al. (2018) proposed a big data analysis based secure cluster management architecture for the optimized control plane.

For dynamics, Peng et al. (2014) formulated the relationship between executor attack surfaces and time. The conclusion was that executor attack surfaces would rise quickly with the time going on. So, shortening the life cycle of executors is an effective way to reduce attack surface sizes. Based on this idea, Guo and Bhattacharya (2014) presented the dynamic switching of the task execution environment, which

can bring a considerable security gain, especially when there is a huge difference in the task execution environment before and after environment switching. However, how to choose the moment of environment switching is a difficult question.

### 3 Security risks of cloud workflows

Normally, a workflow is modeled by the directed acyclic graph (DAG), which can be represented by  $G=(V, E)$ , where  $V=\{v_1, v_2, \dots, v_n\}$  denotes sub-tasks in the workflow and  $E$  denotes dependent data between sub-tasks. For example,  $e_{i,j} \in E$  denotes dependent data generated by  $v_i$  and consumed by  $v_j$ .

In cloud workflow systems, each sub-task in the workflow needs to be allocated to VMs to be executed. Cloud computing platforms adopt virtualization technology to provide flexible resource allocation for workflow execution, but multi-tenant coexistence service mode brings many security threats. For example, adversaries can bypass logical isolation between VMs via side channels to attack VMs performing the workflow (Verma et al., 2017). Attackers can choose to interrupt the execution of the workflow or tamper with the execution results of the workflow. Except for the attacks against VMs, the hypervisor deployed in computing nodes is also insecure. A kind of vulnerabilities called "VM escape" can be employed to help attackers access the privilege domain of the hypervisor, and then all the VMs supported by this hypervisor can be compromised easily (Grobauer et al., 2011). Furthermore, some experienced attackers are able to launch a basic input/output system (BIOS) attack and directly threaten the security of physical servers in the cloud environment (Stewin and Bystrov, 2012; Kallenberg et al., 2013).

Security risks of cloud workflow exist mainly in two aspects. First, sub-tasks in the same workflow are dependent, so any sub-task interruption will terminate the workflow execution. A feasible method is that each sub-task is submitted to multiple task executors to perform. However, from the perspective of the attack surface, the size of a common attack surface of multiple executors will not always reduce with the increase of the number of redundant executors. It is supposed that there are  $M$  task executors,  $e_1, e_2, \dots, e_M$ , whose attack surfaces are represented by  $f(e_1)$ ,

$f(e_2), \dots, f(e_M)$  respectively, and the common attack surface  $f(e_1, e_2, \dots, e_M)$  is defined as

$$f(e_1, e_2, \dots, e_M) = f(e_1) \cap f(e_2) \cap \dots \cap f(e_M). \quad (1)$$

Therefore, multiple homogeneous task executors cannot improve the security of workflow execution significantly. Second, cloud workflows always take a long time to be executed, providing sufficient preparation and attack time for adversaries.

### 4 Mimic cloud workflow execution system

#### 4.1 Framework of the mimic cloud workflow execution system

Important symbols used in Section 4 are listed in Table 1.

The mimic cloud workflow system consists of a controller and multiple computing nodes (Fig. 1). The controller is responsible mainly for four aspects of works: workflow analysis, resource management, task scheduling, and workflow monitoring. When the controller receives a cloud workflow execution request, the workflow analyzer will analyze the structure of the submitted workflow and determine

the task execution sequence. The resource manager is responsible for providing virtual computing resources for workflow execution. The task scheduler is in charge of selecting suitable virtual resources for the task that needs to be performed currently. The workflow monitor collects the status information of workflow execution in real time. When the monitor finds that the current sub-task has been finished, the monitor will send a message to the workflow analyzer to select the next sub-task to be executed.

Table 1 Symbol description

Symbol	Description
$o$	Type of operating system
$v(o_x)$	Number of vulnerabilities of $o_x$
$v(o_1, o_2, \dots, o_n)$	Number of common vulnerabilities among $o_1, o_2, \dots, o_n$
$f(o_1, o_2, \dots, o_n)$	Common attack surface among $v(o_1, o_2, \dots, o_n)$
$a$	Attack strategy
$P(o_j a_i)$	Probability of successful attacks when the attack strategy is $a_i$ and the OS of the attacked target is $o_j$
$h$	Hypervisor type
$s$	Physical server type
$r$	Region
$\lceil \cdot \rceil$	Operator that rounds up to an integer

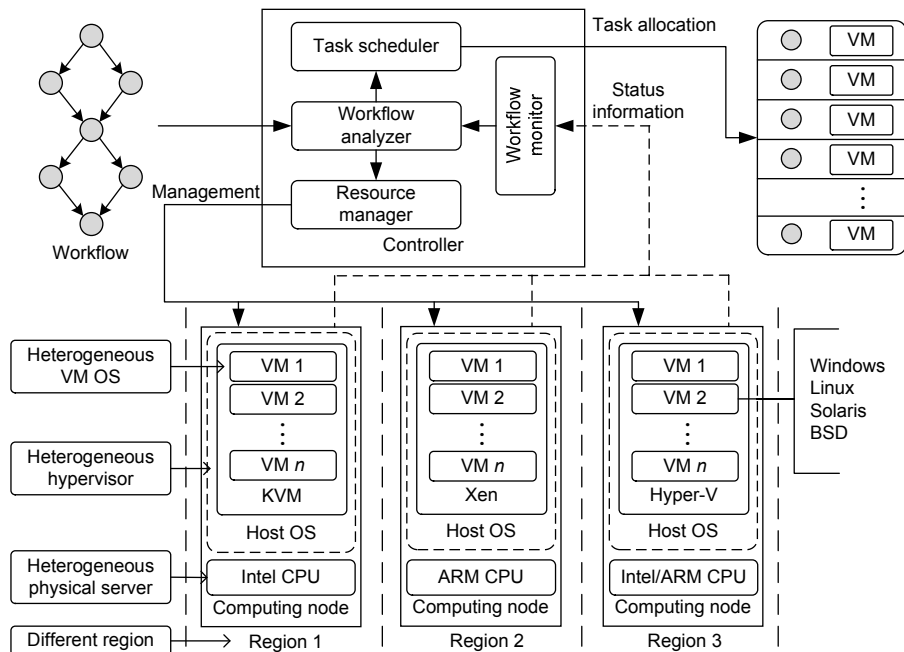


Fig. 1 Framework of the mimic cloud workflow execution system

In Section 3, we have discussed that there are a variety of threats that can threaten the security of VMs, hypervisors, and hardware in cloud systems. For these problems, many scholars have proposed to employ the redundancy method to protect the components in the cloud system. However, homogeneous redundant components cannot achieve good defense effects due to the same vulnerabilities and structures. Inspired by the idea of heterogeneity in mimic defense, we build heterogeneous computing nodes to undertake workflow execution, which includes heterogeneous VM OSs, hypervisors, and physical servers. Heterogeneous VM OSs contain Windows Server 2003, Windows Server 2008, Windows Server 2012, Ubuntu, Debian, Redhat, Solaris, OpenSolaris, OpenBSD, NetBSD, and FreeBSD. Heterogeneous hypervisors contain KVM, Xen, and Hyper-V. Heterogeneous physical servers contain Intel servers and ARM servers.

#### 4.2 Heterogeneous task executor cluster

In cloud workflow systems, each sub-task in the workflow should be scheduled in the corresponding virtual resource to be executed, and the virtual resource which is responsible for sub-task execution is called the task executor. In many cloud workflow systems, each sub-task in the workflow is executed by a single task executor. It is very dangerous because the attackers can interrupt the workflow execution or tamper with the workflow data to cause erroneous execution results. Ding et al. (2017) built a replica for

each sub-task in the workflow, and then the two copies of the same sub-task would be scheduled in different VMs to execute; that is, each sub-task in the workflow would be executed by two task executors. However, the security of workflow execution does not depend on the number of executors, but on the size of the attack surface of executors. If the redundant executors are homogeneous, the size of the attack surface will not decrease with the increase of the number of task executors. Therefore, we use heterogeneous task executors to reduce the size of a common attack surface of task executors and employ the number of vulnerabilities to quantize the size of the common attack surface of heterogeneous task executors. The heterogeneity of task executors is reflected in the heterogeneity of physical servers, hypervisors, and VMs. However, due to the lack of information about the vulnerabilities of physical servers and hypervisors, we can quantize only the size of the common attack surface of heterogeneous VMs.

Taking 11 OSs as examples, i.e., OpenBSD (OB), NetBSD (NB), FreeBSD (FB), Windows Server 2003 (W03), Windows Server 2008 (W08), Windows Server 2012 (W12), Ubuntu (U), Debian (D), Redhat (R), Solaris (Sol), and OpenSolaris (OSol), we record the vulnerabilities of these OSs and the common vulnerabilities between them according to the data published by common vulnerabilities and exposures (CVE) (Table 2), which can be used to measure the heterogeneity between OSs.

The number of common vulnerabilities between

**Table 2 Statistics on the number of vulnerabilities in each operating system and that between different operating systems**

Operating system	Number of vulnerabilities										
	OB	NB	FB	W03	W08	W12	U	D	R	OSol	Sol
OB	<b>149</b>	45	58	2	1	0	3	2	10	13	1
NB	45	<b>139</b>	56	1	1	0	0	4	8	16	0
FB	58	56	<b>284</b>	3	2	0	6	9	22	22	0
W03	2	1	3	<b>412</b>	344	85	0	0	1	7	0
W08	1	1	2	344	<b>846</b>	385	0	0	0	0	0
W12	0	0	0	85	385	<b>467</b>	0	0	0	0	0
U	3	0	6	0	0	0	<b>840</b>	303	179	86	1
D	2	4	9	0	0	0	303	<b>995</b>	220	73	1
R	10	8	22	1	0	0	179	220	<b>1518</b>	69	1
OSol	13	16	22	7	0	0	86	73	69	<b>365</b>	27
Sol	1	0	0	0	0	0	1	1	1	27	<b>100</b>

OB: OpenBSD; NB: NetBSD; FB: FreeBSD; W03: Windows Server 2003; W08: Windows Server 2008; W12: Windows Server 2012; U: Ubuntu; D: Debian; R: Redhat; Sol: Solaris; OSol: OpenSolaris. The number of vulnerabilities in each operating system is in boldface

$o_k$  and  $o_l$  is denoted by  $v(o_k, o_l)$ , and  $f(o_k, o_l)$  representing the size of the common attack surface between  $o_k$  and  $o_l$  is defined as

$$f(o_k, o_l) = k_1 v(o_k, o_l), \quad (2)$$

where  $k_1 > 0$  is a constant. We can continue to use this approach to represent the common attack surface among more than two different OSs. For example,  $f(o_1, o_2, \dots, o_n)$  representing the common attack surface among  $o_1, o_2, \dots, o_n$  can be calculated by

$$f(o_1, o_2, \dots, o_n) = k_2 v(o_1, o_2, \dots, o_n), \quad (3)$$

where  $k_2 > 0$  is a constant. However, the data of  $v(o_1, o_2, \dots, o_n)$  is difficult to obtain. We adopt another way to represent  $f(o_1, o_2, \dots, o_n)$ , which is

$$f(o_1, o_2, \dots, o_n) = \sum_{1 \leq i < j \leq n} f(o_i, o_j) = k_1 \sum_{1 \leq i < j \leq n} v(o_i, o_j). \quad (4)$$

To reduce  $f(o_1, o_2, \dots, o_n)$ , the number of common vulnerabilities between any two OSs should be minimized. Therefore, we propose the maximum heterogeneity OS selection algorithm to build the VM cluster with the smallest common attack surface, and the steps of the algorithm are described as follows:

1. It is assumed that there are  $m$  kinds of candidate OSs in the image library, which is denoted by  $\mathbf{O}=(o_1, o_2, \dots, o_m)$ . In the OS selection algorithm, the first OS  $o_s$  ( $1 \leq s \leq m$ ) is selected randomly, and  $\mathbf{V}$  representing the number of common vulnerabilities between OSs is initialized by

$$\mathbf{V} = (\overbrace{0, 0, \dots, 0}^m). \quad (5)$$

2. Use  $\mathbf{V}_s=(v_1^s, v_2^s, \dots, v_m^s)$  to represent the number of common vulnerabilities between  $o_s$  and other OSs. If  $o_s$  is Ubuntu,  $\mathbf{V}_s=(3, 0, 6, 0, 0, 0, 840, 303, 179, 86, 1)$  according to Table 2. Then  $\mathbf{V}$  is updated as

$$\mathbf{V} + \mathbf{V}_s \rightarrow \mathbf{V} = (v_1, v_2, \dots, v_m). \quad (6)$$

3. The next OS  $o_s$  will be selected based on the following condition:

$$s \in \{g | v_g = \min(v_1, v_2, \dots, v_m)\}. \quad (7)$$

4. If the number of VMs does not meet the requirements, return to step 2; otherwise, the algorithm is over.

In addition to considering the diversity of OSs, when constructing heterogeneous task executors, the differences between the hypervisors, physical servers, and regions should be ensured as much as possible.

### 4.3 Delayed decision mechanism

Due to the low common attack surface among heterogeneous task executors, it is very difficult for adversaries to compromise all the executors, but it is possible to compromise some of them. In this situation, multiple different results will be generated in the executor cluster. To ensure that the executor cluster can produce consistent results, we deploy the result decision module to perform the vote mechanism.

It is assumed that there are  $K$  executors in the executor cluster. If the decision module has received at least  $\lceil K/2 \rceil$  consistent results, the current sub-task execution is valid; otherwise, it is invalid. However, the time for each executor to complete the same sub-task is not the same. Let  $t_f$  and  $t_l$  denote the time when the first and the last results are generated, respectively. If the decision module outputs only the consistent result after receiving all of the results generated by executors, the execution time of each sub-task will increase by  $t_l - t_f$ , which will waste much time. Therefore, the delayed decision mechanism is proposed. As shown in Fig. 2, it is supposed that executor 1 is the first to finish the execution of sub-task  $X$  with the generated result  $A$ . Then result  $A$  is used directly for the execution of sub-task  $Y$ . After receiving the output of executor 1, the decision module will wait for some time to collect outputs of the other executors. It is assumed that the executor not producing a result within the specified time is also a form of output. The decision module will encounter four cases during the collection of outputs. Case 1: The results of all executors are the same. In this case, the system will not take any action. Case 2: More than half of the results are the same, and then the decision module will regard this result as the final result. However, the final result is different from the result of executor 1. In this case, the system will take two actions. First, sub-task  $X$  will be re-executed when there are some leisure resources. Second, sub-task  $Y$  will be re-executed using the final result as the input. Case 3:

More than half of the results are the same, and then the decision module will regard this result as the final result. The final result is the same as the result of executor 1. In this case, sub-task  $X$  will be re-executed when there are some leisure resources. Case 4: The number of identical results is smaller than half of the number of executors. In this case, the system will take two actions. First, sub-task  $X$  will be re-executed immediately. Second, sub-task  $Y$  will be terminated. Based on this method, the workflow execution will go wrong only if all executors are infiltrated by the attacker and output the same error result. The delayed decision mechanism can effectively improve the reliability and credibility of workflow execution without reducing the efficiency significantly.

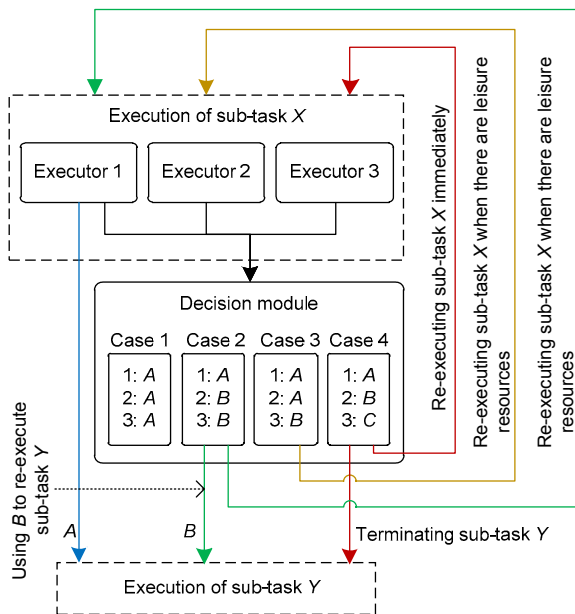


Fig. 2 Delayed decision mechanism

#### 4.4 Elastic executor generation and recycling mechanism

In Section 4.2, it has been proposed to execute each sub-task of the workflow by multiple heterogeneous executors; in this way, multiple sub-task copies replace the original single sub-task (Fig. 3). The workflow whose sub-tasks are replaced by multiple sub-task copies can be divided into multiple independent workflows. In cloud computing platforms, the bandwidth of intra-domain transmission is much higher than that of inter-domain transmission. So,

considering the fault tolerance and efficiency, sub-tasks of the same workflow should be placed into the executors located in the same region, and different workflows should be placed into different regions.

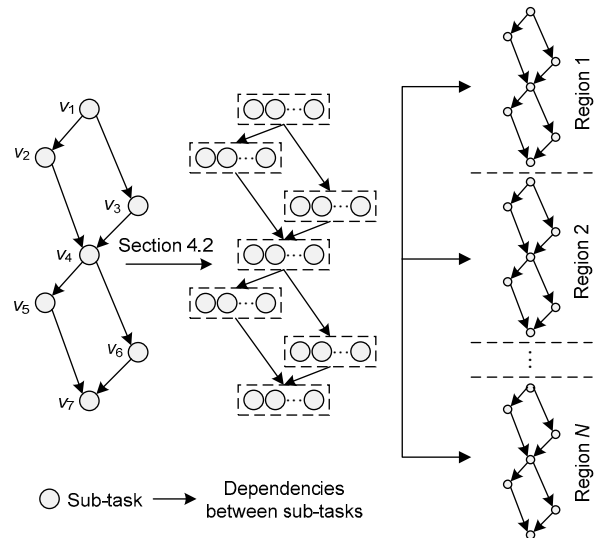


Fig. 3 Workflow whose sub-tasks are replaced by multiple sub-task copies being divided into multiple independent workflows

It is supposed that the number of divided workflows is  $K$ , and each divided workflow will be compromised by adversaries with the probabilities of  $p_1, p_2, \dots, p_K$  respectively. The total execution of the workflow will fail with the probability of  $P$ , and  $P \propto p_i$  ( $1 \leq i \leq K$ ). A feasible method to reduce  $P$  is reducing the value of  $p_i$ . Therefore, we present the elastic executor generation and recycling mechanism, which can improve the security of each divided workflow by shortening the life cycle of executors (Peng et al., 2014).

Compared with a traditional distributed computing system, such as grid computing and cluster computing, cloud computing has the advantage of flexible resource management. Computing resources in the cloud computing platform exist in the form of virtualization, which can realize elastic resource deployment and recycling. In the elastic executor generation and recycling strategy, task executors having finished the sub-task execution will be destroyed, and the reclaimed resource is used for deploying new task executors to perform the next sub-task. According to the theory of cyber kill chain (Yadav and Rao, 2015), a full attack consists of seven steps: reconnaissance,

weaponization, delivery, exploitation, installation, command, and control objectives. It is assumed that it will take time  $t_a$  for adversaries to launch a full attack. If the attacked executors have finished task execution before  $t_a$  and the next task is scheduled to a newly generated executor, adversaries cannot launch a full attack. Furthermore, since each newly generated executor is in a clean state, the elastic executor generation and recycling mechanism can act as the cleaning mechanism, reducing the risks of executors being implanted in the back door or virus.

#### 4.5 Dynamic workflow execution environment switching strategy

The structure of heterogeneous task executors realizes execution environment diversity on the spatial axis, while the workflow execution environment switching strategy achieves execution environment diversity on the timeline. Normally, computing nodes of the cloud platform in the same region are homogeneous. Therefore, in the workflow execution environment switching strategy, we consider only the OS divergence and ignore the divergence of hypervisor types and physical server types.

It is assumed that there is a submitted workflow consisting of a sub-task set  $V(v_1, v_2, \dots, v_m)$ , where  $v_i$  ( $1 \leq i \leq m$ ) represents that it is the  $i^{\text{th}}$  sub-task in the execution order. According to Section 4.2, each sub-task  $v_i$  will be copied as multiple replicas and executed in different regions. Suppose that there are  $k$  regions  $R(r_1, r_2, \dots, r_k)$ , where workflows are executed in parallel,  $v_i^j$  ( $1 \leq i \leq m, 1 \leq j \leq k$ ) denotes the replica of  $v_i \in V$  executed in region  $r_j \in R$ , and  $o_i^j$  denotes the OS of the executor in region  $r_j$  which performs sub-task  $v_i$ .

The workflow execution environment switching strategy is shown in Fig. 4.  $o_1^j$  can be selected by the OS selection algorithm with the maximum heterogeneity.  $o_2^1$  should have the smallest common attack surface with  $o_1^1$ , and  $o_2^2$  should have the smallest common attack surface with both  $o_1^1$  and  $o_1^2$ . Therefore, for  $o_a^b$  ( $1 \leq a \leq m, 1 < b \leq k$ ), it should have the smallest common attack surface with  $\{o_a^e | 1 \leq e \leq b-1\}$  and  $o_{a-1}^b$ . Based on this idea, the workflow execution environment switching strategy is shown in

Algorithm 1, which can achieve the purpose of confusing attackers.

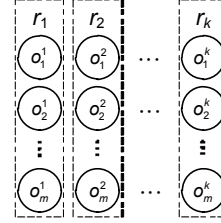


Fig. 4 Illustration of the workflow execution environment switching strategy

---

#### Algorithm 1 Workflow execution environment switching strategy

---

**Input:** number of task executors  $N$ , diverse OS types  $O[m_o] = \{o_1, o_2, \dots, o_{m_o}\}$ , and sub-tasks in the workflow  $V[m][k] = \{v_i^j | 1 \leq i \leq m, 1 \leq j \leq k\}$  where  $V[1][2] = v_1^2$

**Output:** OSs of executors performing each sub-task  $E[m][k] = \{o_i^j | 1 \leq i \leq m, 1 \leq j \leq k\}$  where  $E[1][2] = o_1^2$

```

1  for  $i=0; i < m; i++$ 
2  for  $j=0; j < k; j++$ 
3  if  $i=0 \ \&\& \ j=0$ 
4     $E[i][j]$ =selecting an OS from  $O[\cdot]$  randomly
5  else if  $i=0 \ \&\& \ j! = 0$ 
6     $E[i][j]$ =selecting the OS having the smallest attack
    surface with  $E' = \{E[i][d] | 0 \leq d \leq j-1\}$  by the OS
    selection algorithm with the maximum
    heterogeneity
7  else if  $i! = 0 \ \&\& \ j=0$ 
8     $E[i][j]$ =selecting the OS having the smallest attack
    surface with  $E[i-1][j]$  by the OS selection
    algorithm with the maximum heterogeneity
9  else
10    $E[i][j]$ =selecting the OS having the smallest attack
    surface with  $E' = \{E[i][d] | 0 \leq d \leq j-1\}$  and  $E[i-1][j]$ 
    by the OS selection algorithm with the maximum
    heterogeneity
11  end if
12  end for
13  end for
14  return  $E[m][k]$ 

```

---

## 5 Experiment and discussion

### 5.1 Security evaluation of the heterogeneous task executor cluster

First, we will evaluate the impact of the number of task executors on system security. It is assumed that there are  $m$  types of candidate OSs in the image library, denoted by  $o_1, o_2, \dots, o_m$ . Adversaries know

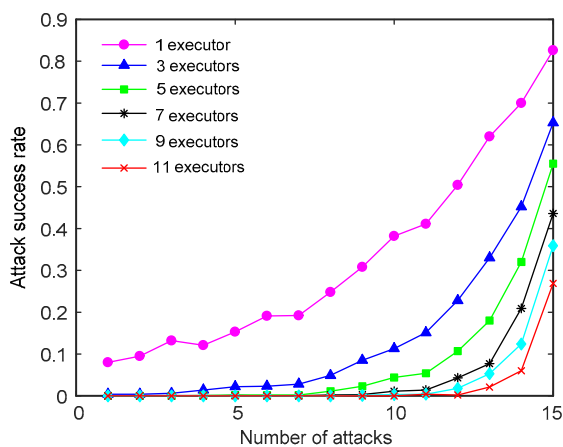


that the OS of each VM must belong to one of  $o_1, o_2, \dots, o_m$ , but cannot accurately determine which one it is. Adversaries will randomly select an OS as the attack strategy each time. Let  $a_i$  represent the attack strategy against  $o_i$ . It is supposed that attackers adopt strategy  $a_i$  to attack the task executor cluster, and the executor whose OS is  $o_i$  will be compromised with probability 1 and the executor whose OS is  $o_j$  ( $j \neq i$ ) will be compromised with probability  $p$ :

$$p = P(o_j|a_i) = \frac{f(o_j, o_i)}{f(o_i, o_i)} = \frac{v(o_j, o_i)}{v(o_i)}. \quad (8)$$

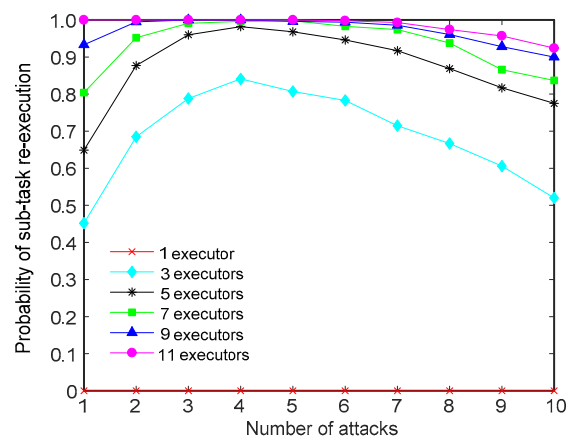
Note that the defined attack model is an extreme case, which is convenient for analyzing the general law between system parameters and system security. In a practical situation, successful attacks depend on a lot of factors. Even if adversaries know the system type of target, it is difficult to compromise the target by one attack. Therefore, in the actual environment, the security of a mimic cloud workflow system is much higher than the simulation results.

Based on the attack model, we use the number of attacks and the number of task executors as variables to simulate the attack and defense against cloud workflows. To analyze the impact of executor heterogeneity on system security, the 11 OSs listed in Table 1 are used for tests, and the maximum heterogeneity OS selection algorithm is used for selecting OS types. If at least  $\lceil K/2 \rceil$  executors are compromised, the attack is declared successful. Test results are shown in Fig. 5.



**Fig. 5 Relationship between the number of executors and the attack success rate**

From Fig. 5, we can find that with the increase of the number of attacks, the attack success rate shows a significant upward trend. The number of executors will also affect the system security, since if more executors are contained in the cluster, attackers need to compromise more executors to succeed, which will increase the difficulty of successful attacks. We may find that the larger the number of heterogeneous executors, the better, but this is not the case. Fig. 6 shows the relationship between the probability of sub-task re-execution and the number of executors. From Fig. 6, we can find that the increase of the number of executors will notably increase the probability of sub-task re-execution. In the proposed system, if inconsistent results appear in the executor cluster, the sub-task will be re-executed. The larger the number of executors, the larger the probability that attackers will compromise one of the executors. When the number of attacks exceeds a certain value, the sub-task re-execution probability will show a downward trend, because of the increase of the probability of the erroneous consistent result produced in the executor cluster. Therefore, the increase in the number of executors will effectively reduce the probability that the system produces erroneous results, but at the same time, the probability of sub-task re-execution will be increased, which will cost extra resources.



**Fig. 6 Relationship between the number of executors and the probability of sub-task re-execution**

Then we evaluate the security gain brought by the maximum heterogeneity OS selection algorithm. Except for the maximum heterogeneity OS selection algorithm, the random OS selection algorithm and homogeneous OS selection algorithm are used for

comparison. In the random OS selection algorithm, the OS of each executor is chosen randomly from the image library, and in the homogeneous OS selection algorithm, the OSs of all executors are the same. In this evaluation, the number of executors is three, and we use the number of attacks as the variable to simulate the attack and defense against cloud workflows. If all the executors are compromised, adversaries will win; otherwise, defenders will win. Results are shown in Fig. 7.

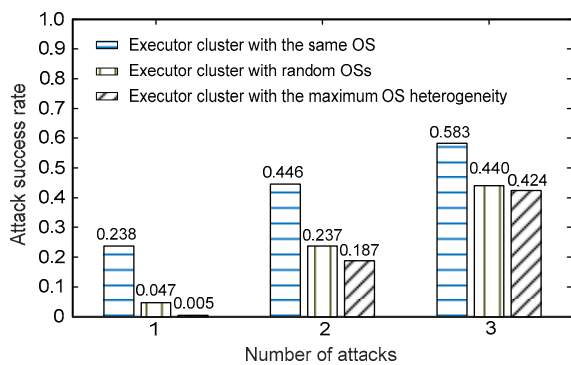


Fig. 7 Security evaluation results of three executor cluster deployment methods

From Fig. 7, we can find that the maximum heterogeneity OS selection algorithm can strengthen the security of the executor cluster. When building an executor cluster with the maximum OS heterogeneity, the small common attack surface makes it difficult for adversaries to compromise multiple executors with one attack. However, with the increase of the number of attacks, the effect of the maximum heterogeneity OS selection algorithm is decreasing.

In this study, 11 OSs have been used to illustrate the impact of executor heterogeneity on system security. When building a practical mimic cloud workflow execution system, two to four common OSs with high heterogeneity are enough.

## 5.2 Security evaluation of elastic executor generation and recycling mechanism

Based on the results of Section 5.1, we conduct the security evaluation of elastic executor generation and recycling mechanism. Define three kinds of attacks, random attack mode without memory (attack R), random attack mode with memory (attack M), and sniffing attack mode (attack S). Attack R: Adversaries will choose the attack strategy from the strategy set

with an equal probability when attacking the executor cluster. Attack M: This mode will record the strategy that has been used by adversaries, and when launching the next attack to the executor cluster, adversaries will select the attack strategy from the unused strategy set with an equal probability. Attack S: Adversaries will choose one of the executors from the executor cluster to attack, no matter whether the launched attack is successful or not, and adversaries can obtain the OS type of this target. The probability of successful attacks is calculated using Eq. (8).

It is assumed that the workflow used for test consists of 10 sub-tasks, and each sub-task is executed in parallel by three executors with the maximum heterogeneity. The time required for each sub-task execution is randomly assigned, but the total time must be 100 h. Two kinds of workflow systems are used for test, one system adopting the elastic executor generation and recycling strategy, but scheduling candidate executors being homogeneous (system E), and the other system not adopting the dynamic executor generation and recycling strategy (system N). Attacks R, M, and S are employed to test the security of these systems. If all executors performing a sub-task are compromised by adversaries, the sub-task is compromised. As long as one sub-task is compromised, it means that the attack against the workflow is successful. Fig. 8 shows the test results.

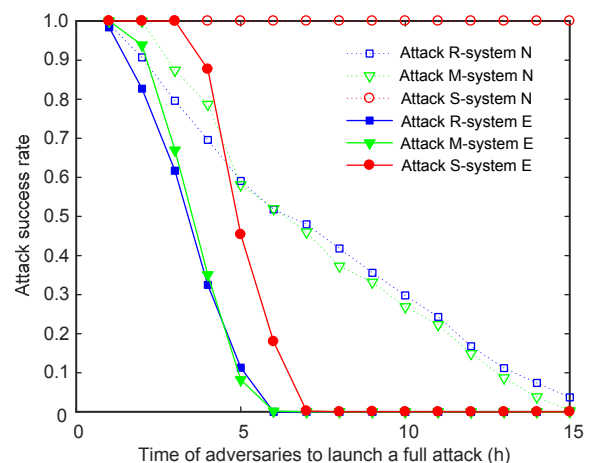
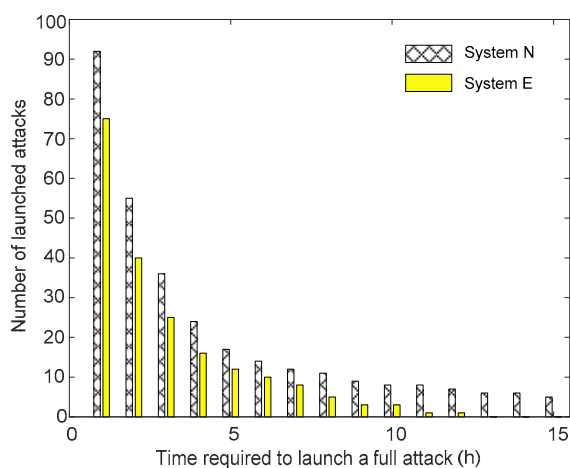


Fig. 8 Test results of security gains brought by the elastic executor generation and recycling strategy

We can find that the shorter the time required for adversaries to launch a full attack, the higher the probability that the system will be compromised. The

distribution of solid and dashed lines in Fig. 8 illustrates that compared with system N, system E is more secure, which proves that the elastic executor generation and recycling mechanism can strengthen the system security. To analyze the reason, we record the number of launched full attacks against the two systems. Fig. 9 illustrates that the dynamic executor generation and recycling strategy can reduce the number of received attacks. If the executor completes task execution before the attacker invades, executor recycling mechanism can effectively interrupt attackers' invasion.



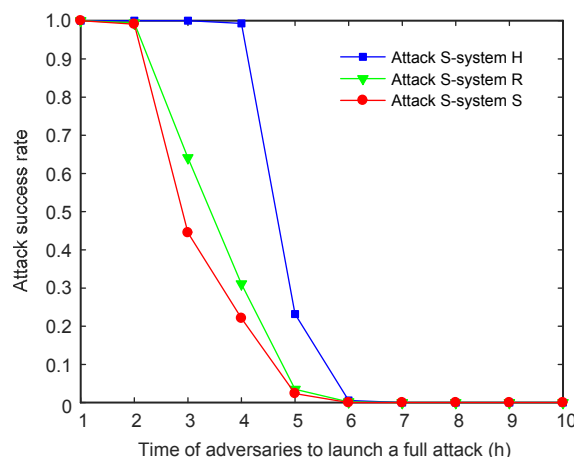
**Fig. 9 Comparison of the number of launched full attacks against the two systems**

### 5.3 Security evaluation of the dynamic workflow execution environment switching strategy

Based on the heterogeneous executor cluster and elastic executor generation and recycling mechanism, we further evaluate the security of the dynamic workflow execution environment switching strategy. In this experiment, the network attack is also simulated by Eq. (8). Each sub-task is processed in parallel by three executors with the maximum heterogeneity.

To compare security gains brought by the dynamic workflow execution environment switching strategy, we construct three kinds of systems. All of these systems adopt the dynamic executor generation and recycling strategy. However, in the first system (system H), executors are homogeneous before and after task scheduling. In the second system (system R), executors are selected randomly before and after task scheduling. In the last system (system S), executors

have the smallest common attack surface before and after task scheduling. Attack S is used to simulate attacks against workflows, and test results are shown in Fig. 10. We can find that under attack S, system S is more secure than systems R and H. Because attack S can detect the OS of a target, attackers can compromise an executor by just two attacks. In system H, OSs of executors before and after scheduling are the same, so attack S can easily compromise system H. Systems R and S can achieve workflow execution environment change, but only system S can ensure that executors before and after scheduling have the smallest common surface, which can effectively reduce the effects of attack S.



**Fig. 10 Test results of security gains brought by the workflow execution environment switching strategy**

### 5.4 White box test of mimic cloud workflow execution system

A white box test environment is built for the mimic cloud workflow execution system based on open-source software OpenStack and OpenDaylight. The system contains a control node (48-core processor, 32-GB memory, 2-TB storage space), four computing nodes (48-core processor, 32-GB memory, 2-TB storage space), and a storage node (48-core processor, 32-GB memory, 4-TB storage space). This system includes three kinds of networks: management network, internal data network, and external data network. The management network is responsible mainly for the transmission of system management commands. The internal data network is primarily responsible for the transfer of dependent data between workflow sub-tasks. The external data network

(192.168.108.0/24) is mainly for communication between VMs and the Internet. In this test, we will simulate the attack scenario that an attacker infiltrates into the VMs through the external data network to interfere with the workflow execution.

This test is verified by an actual workflow (target detection classifier training). The structure of the workflow is shown in Fig. 11. A 20-GB image set is selected as the input from the target detection standard test set ILSVRC2012.

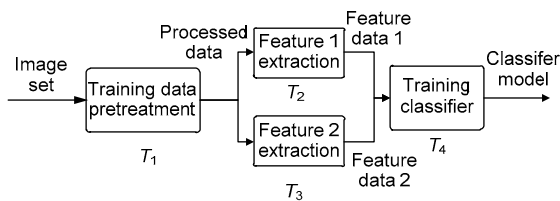


Fig. 11 Actual workflow used for the white box test

In the proposed system, users need to define workflows in the XML format (Chen and Deelman, 2012), and then submit the XML file and input data to the mimic cloud workflow execution system. In this system, the task executor is generated and recycled dynamically, so users need to pre-package the software into the images and upload them to the system image library. In this test, the executive programs are deployed corresponding to each sub-task in Windows Server 2012, Ubuntu 16.04, and CentOS 7.3, and

packaged into images. In addition to submitting workflow definition files and input files, users need to request a certain amount of vCPU, memory, and storage space from the system, and specify the VM configuration. In this test, we apply for 20 vCPU, 20-GB memory, and 1500-GB storage space. The specified VM configuration is 4 vCPU, 4-GB memory, and 300-GB storage space. During the workflow execution process, VMs are generated and recycled dynamically according to the task execution requirements. However, the total configuration of VMs which are running cannot exceed the sum of resources applied by users.

Two sets of experiments are set up to compare the effects.

1. Experiment where there is no attacker invasion

A part of log information is shown in Fig. 12. The system requires users specify the task execution order in the workflow definition file. If each sub-task execution time is known, the heterogeneous earliest finish time (HEFT) algorithm (Topcuoglu et al., 2002) can be used to determine the task execution order. After receiving the workflow execution request, the system will execute three independent workflows in parallel to verify the execution result. Mimic-ResourceManager builds three heterogeneous executors through Nova-Api, and the images used are

```

root@controller:~/home/yyw# MimicWorkflow -xml ./workflow.xml -data ./image -cpu 20 -memory 20 -storage 1500 -vm_cpu 4 -vm_memory 4
-vm_storage 300
total resources: CPU 20 Memory 20 Storage 1500
*****Start workflow resolution*****
T1->T2->T3->T4
*****finish workflow resolution*****
Generate VM, ID: b08ca6a2ca, network: net1, Float IP: 192.168.108.101, image: windows_T1.qcow2, host: server1
Remaining resources: CPU 16 Memory 16 Storage 1200
Generate VM, ID: bb5a7866af, network: net2, Float IP: 192.168.108.104, image: ubuntu_T1.qcow2, host: server2
Remaining resources: CPU 12 Memory 12 Storage 900
Generate VM, ID: 5494c68fd7, network: net3, Float IP: 192.168.108.105, image: centos_T1.qcow2, host: server3
Remaining resources: CPU 8 Memory 8 Storage 600
*****Start T1 Execution*****
VMb08ca6a2ca starts to execute T1
VMbb5a7866af starts to execute T1
VM5494c68fd7 starts to execute T1
VM5494c68fd7 finishes T1, execution result is 68D92759629781090F395093C33C158D
Generate VM, ID: 9b67ff1f2e, network: net3, Float IP: 192.168.108.108, image: ubuntu_T2.qcow2, host: server3
Remaining resources: CPU 4 Memory 4 Storage 300
Generate VM, ID: cdbf5dd657, network: net3, Float IP: 192.168.108.109, image: windows_T3.qcow2, host: server3
Remaining resources: CPU 0 Memory 0 Storage 0
Start data transfer VM5494c68fd7->VM9b67ff1f2e
Start data transfer VM5494c68fd7->VMcdbf5dd657
VMb08ca6a2ca finishes T1, execution result is 68D92759629781090F395093C33C158D
VMbb5a7866af finishes T1, execution result is 68D92759629781090F395093C33C158D
Decision Module: final result of T1: 68D92759629781090F395093C33C158D
*****Finish T1 Execution*****
Finish data transfer VM5494c68fd7->VM9b67ff1f2e
Finish data transfer VM5494c68fd7->VMcdbf5dd657
Recycle VM5494c68fd7
Remaining resources: CPU 4 Memory 4 Storage 300
*****Start T2 Execution*****
VM9b67ff1f2e starts to execute T2
*****Start T3 Execution*****
VMcdbf5dd657 starts to execute T3
*****
Workflow execution result: 1240BB01AA3306CF8D0C0944A58882BF0
Workflow execution time: 2:17:03

```

Fig. 12 A part of the log information about the experiment where there is no attacker invasion (References to color refer to the online version of this figure)

windows\_T1.qcow2, ubuntu\_T1.qcow2, and centos\_T1.qcow2, respectively. To enhance the mutual isolation of heterogeneous executors, we modify the scheduling mechanism of Nova-Scheduler so that the executors belonging to the same workflow are placed as concentrated as possible, as indicated by the red lines in Fig. 12. Furthermore, the executors belonging to the same workflow are placed in the same network, as indicated by the green lines in Fig. 12. In this test, VM5494cb8fd7 takes the lead in completing  $T_1$  task execution, and the md5 value of the execution result is 68D92759629781090F395093C33C158D. Since the delayed decision mechanism is adopted, the next sub-task execution can be performed directly without waiting for the other two executors' results. After VM5494cb8fd7 completes the data transfer with the newly generated VM, the executor recycling mechanism is triggered to destroy the VM.

2. Experiment where a VM is penetrated by attackers

A part of the log information is shown in Fig. 13. It is assumed that attackers know the float IP and password of the VM performing  $T_1$  first time. In this way, attackers can penetrate into the VM through the secure shell (SSH) and tamper with the data, so that the execution result is different from those of the other VMs, as indicated by the red lines in Fig. 13. In this experiment, VM073140da28 which outputs the wrong result is the first to complete  $T_1$ . Due to the delayed decision mechanism, VM073140da28

will directly execute the next sub-task. After VM9f9854a312 and VMf8e9ff30c0 complete  $T_1$  execution, the decision module determines that the execution result of VM073140da28 is incorrect, so the decision module will interrupt the data transmission of VM073140da28 and recycle it. At this moment, the VM on host server 2 does not have the correct  $T_1$  execution result, so the task execution cannot be continued. Since the workflows cannot communicate directly with each other, the control node needs to perform data forwarding, as indicated by the blue line in Fig. 13. Although  $T_1$  has already been executed, since the results of three executors are not agreed, sub-task  $T_1$  will be re-executed when there are idle resources in the system, as indicated by the green line in Fig. 13.

## 6 Conclusions

The multi-tenant coexistence service mode in the cloud computing platform introduces serious security risks. To achieve highly available, reliable, and trusted workflow execution, we have presented the mimic cloud computing task execution system. First, the diversities of physical servers, hypervisors, and operating systems were introduced to build the intrusion-tolerant framework. Based on the framework, common vulnerabilities among different operating systems were used for heterogeneity

```

*****Start T1 Execution*****
VM9f9854a132 starts to execute T1
VM073140da28 starts to execute T1
VMf8e9ff30c0 starts to execute T1
VM073140da28 finishes T1, execution result is AD3B02E08ABBE1D809C5411929CA38BD
Generate VM, ID: 21f6a14ec9, network: net2, float IP: 192.168.108.133, image: windows_T2.qcow2, host: server2
Remaining resources: CPU 4 Memory 4 Storage 300
Generate VM, ID: f925966485, network: net2, float IP: 192.168.108.134, image: centos_T3.qcow2, host: server2
Remaining resources: CPU 0 Memory 0 Storage 0
Start data transfer VM073140da28->VM21f6a14ec9
Start data transfer VM073140da28->VMf925966485
VM9f9854a132 finishes T1, execution result is 68D92759629781090F395093C33C158D
VMf8e9ff30c0 finishes T1, execution result is 68D92759629781090F395093C33C158D
Decision Module: final result of T1: 62686A243C4BB53507816EFD8726D133
Recycle VM073140da28
Remaining resources: CPU 4 Memory 4 Storage 300
Recycle VM21f6a14ec9
Remaining resources: CPU 8 Memory 8 Storage 600
Recycle VMf925966485
Remaining resources: CPU 12 Memory 12 Storage 900
*****
Generate VM, ID: c9a0156fc7, network: net2, float IP: 192.168.108.139, image: centos_T2.qcow2, host: server2
Remaining resources: CPU 0 Memory 0 Storage 0
Start data transfer VM9f9854a132->VMc9a0156fc7
*****
Remaining resources: CPU 4 Memory 4 Storage 300
Generate VM, ID: 9684e91945, network: net1, float IP: 192.168.108.144, image: windows_T1.qcow2, host: server1
Remaining resources: CPU 0 Memory 0 Storage 0
Start data transfer Storage->VM9684e91945
Finish data transfer Storage->VM9684e91945
*****Start T1 Re-Execution*****
VM9684e91945 starts to execute T1
*****
Workflow execution result: 1240BB01AA3D6CF800C8944A58882BF0
Workflow execution time: 2:25:37

```

Fig. 13 A part of the log information about the experiment where a VM is penetrated by attackers (References to color refer to the online version of this figure)

measurement and executor deployment. With the flexible resource management of the cloud computing platform, the elastic executor generation and recycling mechanism was proposed, which can not only shorten the life cycle of executors, but also act as the clean mechanism to keep the pure state of executors. Then inspired by the dynamic thought in mimic defense, the dynamic workflow execution environment switching strategy was presented to confuse the adversaries. Experimental results showed that the proposed system can effectively strengthen the reliability and credibility of cloud workflow execution.

## References

- Ainapure B, Shah D, Rao AA, 2018. Adaptive multilevel fuzzy-based authentication framework to mitigate cache side channel attack in cloud computing. *Int J Model Simul Sci Comput*, 9(5):1850045. <https://doi.org/10.1142/S1793962318500459>
- Aktas MF, Haldeman G, Parashar M, 2014. Flexible scheduling and control of bandwidth and in-transit services for end-to-end application workflows. 4<sup>th</sup> IEEE Int Workshop on Network-Aware Data Management, p.28-31. <https://doi.org/10.1109/NDM.2014.9>
- Casas I, Taheri J, Ranjan R, et al., 2017. A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems. *Fut Gener Comput Syst*, 74: 168-178. <https://doi.org/10.1016/j.future.2015.12.005>
- Chen WW, Deelman E, 2012. Workflowsim: a toolkit for simulating scientific workflows in distributed environments. 8<sup>th</sup> IEEE Int Conf on E-Science, p.1-8. <https://doi.org/10.1109/eScience.2012.6404430>
- Deldari A, Naghibzadeh M, Abrishami S, 2017. CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud. *J Supercomput*, 73(2): 756-781. <https://doi.org/10.1007/s11227-016-1789-5>
- Ding YS, Yao GS, Hao KR, 2017. Fault-tolerant elastic scheduling algorithm for workflow in cloud systems. *Inform Sci*, 393:47-65. <https://doi.org/10.1016/j.ins.2017.01.035>
- Evans N, Thompson M, 2016. Multiple operating system rotation environment moving target defense. US Patent, 9 294 504.
- Garcia M, Bessani A, Gashi I, et al., 2011. OS diversity for intrusion tolerance: myth or reality? 41<sup>st</sup> IEEE Int Conf on Dependable Systems & Networks, p.383-394. <https://doi.org/10.1109/DSN.2011.5958251>
- Garcia M, Bessani A, Gashi I, et al., 2014. Analysis of operating system diversity for intrusion tolerance. *Softw Pract Exp*, 44(6):735-770. <https://doi.org/10.1002/spe.2180>
- Grobauer B, Walloschek T, Stocker E, 2011. Understanding cloud computing vulnerabilities. *IEEE Secur Priv*, 9(2): 50-57. <https://doi.org/10.1109/MSP.2010.115>
- Guo MZ, Bhattacharya P, 2014. Diverse virtual replicas for improving intrusion tolerance in cloud. 9<sup>th</sup> Annual Cyber and Information Security Research Conf, p.41-44. <https://doi.org/10.1145/2602087.2602116>
- Gupta I, Kumar MS, Jana PK, 2016. Compute-intensive workflow scheduling in multi-cloud environment. Int Conf on Advances in Computing, Communications and Informatics, p.315-321. <https://doi.org/10.1109/ICACCI.2016.7732066>
- Hu HC, Wang ZP, Cheng GZ, et al., 2017. MNOS: a mimic network operating system for software defined networks. *IET Inform Secur*, 11(6):345-355. <https://doi.org/10.1049/iet-ifs.2017.0085>
- Juve G, Deelman E, 2011. Scientific workflows in the cloud. In: Cafaro M, Aloisio G (Eds.), *Grids, Clouds and Virtualization*. Springer, London, p.71-91. [https://doi.org/10.1007/978-0-85729-049-6\\_4](https://doi.org/10.1007/978-0-85729-049-6_4)
- Kallenberg C, Butterworth J, Kovah X, et al., 2013. Defeating Signed BIOS Enforcement. <https://www.mitre.org/sites/default/files/publications/defeating-signed-bios-enforcement.pdf>
- Lee YC, Han H, Zomaya AY, et al., 2015. Resource-efficient workflow scheduling in clouds. *Knowl-Based Syst*, 80: 153-162. <https://doi.org/10.1016/j.knsys.2015.02.012>
- Lv HW, Lin JY, Wang HQ, et al., 2015. Analyzing the service availability of mobile cloud computing systems by fluid-flow approximation. *Front Inform Technol Electron Eng*, 16(7):553-567. <https://doi.org/10.1631/FITEE.1400410>
- Pandey S, Wu LL, Guru SM, et al., 2010. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. 24<sup>th</sup> IEEE Int Conf on Advanced Information Networking and Applications, p.400-407. <https://doi.org/10.1109/AINA.2010.31>
- Peng W, Li F, Huang CT, et al., 2014. A moving-target defense strategy for Cloud-based services with heterogeneous and dynamic attack surfaces. IEEE Int Conf on Communications, p.804-809. <https://doi.org/10.1109/ICC.2014.6883418>
- Platania M, Obenshain D, Tantillo T, et al., 2014. Towards a practical survivable intrusion tolerant replication system. 33<sup>rd</sup> IEEE Int Symp on Reliable Distributed Systems, p.242-252. <https://doi.org/10.1109/SRDS.2014.16>
- Platania M, Obenshain D, Tantillo T, et al., 2016. On choosing server- or client-side solutions for BFT. *ACM Comput Surv*, 48(4), Article 61. <https://doi.org/10.1145/2886780>
- Stewin P, Bystrov I, 2012. Understanding DMA malware. 9<sup>th</sup> Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment, p.21-41. [https://doi.org/10.1007/978-3-642-37300-8\\_2](https://doi.org/10.1007/978-3-642-37300-8_2)
- Topcuoglu H, Hariri S, Wu MY, 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans Parallel Distrib Syst*, 13(3): 260-274. <https://doi.org/10.1109/71.993206>
- Verma A, Mittal M, Chhabra B, 2017. The mutual authentication scheme to detect virtual side channel attack in cloud computing. *Int J Comput Sci Inform Secur*, 15(3):83-98.

- Wang JW, Korambath P, Altintas I, et al., 2014. Workflow as a service in the cloud: architecture and scheduling algorithms. *Proc Comput Sci*, 29:546-556. <https://doi.org/10.1016/j.procs.2014.05.049>
- Wu J, Dong MX, Ota K, et al., 2018. Big data analysis-based secure cluster management for optimized control plane in software-defined networks. *IEEE Trans Netw Serv Manag*, 15(1):27-38. <https://doi.org/10.1109/TNSM.2018.2799000>
- Yadav T, Rao AM, 2015. Technical aspects of cyber kill chain. 3<sup>rd</sup> Int Symp on Security in Computing and Communication, p.438-452. [https://doi.org/10.1007/978-3-319-22915-7\\_40](https://doi.org/10.1007/978-3-319-22915-7_40)
- Yao GS, Ding YS, Ren LH, et al., 2016. An immune system-inspired rescheduling algorithm for workflow in cloud systems. *Knowl-Based Syst*, 99:39-50. <https://doi.org/10.1016/j.knsys.2016.01.037>
- Yao GS, Ding YS, Hao KR, 2017. Using imbalance characteristic for fault-tolerant workflow scheduling in cloud systems. *IEEE Trans Parall Distrib Syst*, 28(12):3671-3683. <https://doi.org/10.1109/TPDS.2017.2687923>
- Yuan D, Yang Y, Liu X, et al., 2012. A data dependency based strategy for intermediate data storage in scientific cloud workflow systems. *Concurr Comput Pract Exp*, 24(9): 956-976. <https://doi.org/10.1002/cpe.1636>
- Zheng ZB, Zhou TC, Lyu MR, et al., 2012. Component ranking for fault-tolerant cloud applications. *IEEE Trans Serv Comput*, 5(4):540-550. <https://doi.org/10.1109/TSC.2011.42>