Alleviating the Impact of Abnormal Events Through Multi-Constrained VM Placement

Gongming Zhao, *Member, IEEE*, Jiawei Liu, Yutong Zhai, Hongli Xu, *Member, IEEE*, and Huang He, *Member, IEEE*

***Abstract*—As a simple and low-cost way to obtain enough computing resources, more and more tenants migrate their tasks to the cloud. However, the frequent occurrence of abnormal events (e.g., malicious tenants and node failures) in the cloud will seriously affect the tenants' QoS. Conventionally, the cloud vendors reduce the frequency of abnormal events by deploying auxiliary systems, which requires additional costs and increases network complexity. Considering that it is an unrealistic expectation to eliminate the occurrence of abnormal events in clouds, this paper proposes a complementary scheme to alleviate the negative impact scope when an abnormal event occurs through multi-constrained VM placement without consuming additional resources. Specifically, when deploying VMs, we limit the number of pods (or service nodes) each tenant can access and the number of tenants hosted by each pod (or service node). However, the multi-dimensional interaction among numerous system parameters and performance/resource considerations makes the problem of multi-constrained VM placement for alleviating the impact of abnormal events very challenging. To solve this problem, we formulate an integer linear programming and propose a rounding-based algorithm with a logarithmic approximation ratio. We implement our proposed algorithm on a physical testbed. The experimental and simulation results show the high efficiency of the proposed algorithm. For example, our algorithm reduces the impact scope of service node failure by 60%, the impact scope of malicious tenants by 40%, and the tenant task makespan by 25% compared with other alternatives.**

***Index Terms*—Abnormal events, clouds, resource allocation, VM placement.**

I. Introduction

With the development of cloud computing, more and more enterprises migrate their computation tasks to the cloud since such a migration significantly reduces the complexity and costs of managing private data centers. A typical cloud (eg., Amazon EC2 [2] and Alibaba cloud [3]) is composed of many *pods*. In each pod, cloud vendors deploy a large number of *compute nodes* to provide computing resources (eg., CPU) for tenants in the form of virtual machines (VMs). Moreover, the *service node* in each pod can provide various network services for tenants, such as virtual private network (VPN) [4] and elastic load balancing (ELB) [5].

Abnormal events in a multi-tenant cloud are caused by either external or internal factors, which are widespread and will reduce the quality of service (QoS) of tenants [6], [7]. On the one hand, the attack of malicious tenants is the most common abnormal event caused by external factors. Specifically, there are massive tenants in a commodity cloud, and the diversity of tenants means that there may be many malicious tenants [8], [9]. Malicious tenants will launch extensive network attacks, including denial of service (DoS) attacks and co-residency attacks [10] against the service nodes, paralyzing service nodes and decreasing network performance. Moreover, by executing malicious programs on the attacked VMs, a malicious tenant can hijack them so that they can not serve legitimate tenants. According to a recent report [11], malicious tenants access unauthorized VMs more than ten times per minute in a small-size data center. On the other hand, Schroeder et al. [12] observe that most abnormal events caused by internal factors in clouds occur due to node failures. According to [13], a system with 100,000 processors will experience a processor failure every few minutes. In addition, service nodes fail more frequently than compute nodes, the median time between two consecutive failures of firewall, load balancer, and intrusion detection on service nodes is 7.5 hours, 5.2 hours, and 20 minutes, respectively [14]. When a service node is unavailable, the served requests should be rescheduled to other available service nodes, which will cause a long rescheduling delay and decrease tenants' QoS [15].

Conventionally, the cloud vendors cope with abnormal events by deploying accessorial systems, such as an intrusion detection system to protect service nodes against malicious tenants [16] and a monitoring system to monitor the service node status [17]. Although the above methods can reduce the frequency of abnormal events in the cloud, it is an unrealistic expectation to entirely eliminate the occurrence of abnormal events. Alternatively, this work tries to alleviate the negative impact caused by abnormal events through multi-constraint VM placement without consuming additional resources. It is worth noting that existing works

Manuscript received 27 August 2022; revised 15 February 2023; accepted 21 February 2023. Date of publication 24 February 2023; date of current version 16 March 2023. This work was supported in part by the National Science Foundation of China (NSFC) under Grant 62102392, in part by the National Science Foundation of Jiangsu Province under Grant BK20210121, in part by Hefei Municipal Natural Science Foundation under Grant 2022013, and in part by the Youth Innovation Promotion Association of CAS under Grant 2023481. Some preliminary results of this paper were published in the IEEE/ACM International Symposium on Quality of Service (IWQoS) 2021 [DOI: 10.1109/IWQOS52092.2021.9521344]. Recommended for acceptance by M. Si. *(Corresponding author: Hongli Xu.)*

Gongming Zhao, Jiawei Liu, Yutong Zhai, and Hongli Xu are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu 215123, China (e-mail: gmzhao@ustc.edu.cn; liujiawe@mail.ustc.edu.cn; zhaiyutong@meituan.com; xuhongli@ustc.edu.cn).

Huang He is with the School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu 215123, China (e-mail: huangh@suda.edu.cn).

Digital Object Identifier 10.1109/TPDS.2023.3248681

mainly focus on reducing the frequency of abnormal events. We regard that our work plays a *complementary* story to handling abnormal events in the cloud, rather than trying to improve previous methods or even substitute them. Note that, in practice, VM placement (VMP) problems can be generally classified into two categories [18], [19]: offline VMP and online VMP. The former is usually performed when multiple VMs need to be placed (eg., VM updates, virtual private cloud creation, system initialization). On the contrary, the latter is performed to cope with cloud dynamics. In fact, the two complement each other. In this paper, we focus on offline VMP.

To this end, we will introduce two novel constraints when deploying VMs in the cloud, besides purely pursuing load balancing on both compute nodes and service nodes. First, to control the impact scope of malicious tenants, the VMs belonging to the same tenant will be allocated to at most w service nodes/pods. Second, to control the impact scope of service node failures, each service node/pod will host VMs of up to h tenants. Here w and h are two constant parameters configured by the cloud system. The rationale behind the first constraint is that with proper isolation techniques [20], a malicious tenant will only attack the service nodes that provide services for the allocated VMs. Hence the first constraint helps to limit the damage caused by a single malicious tenant. On the other hand, the failure of a service node will result in performance degradation for all the tenants in the corresponding pods, thus we have the second constraint to limit the impact scope caused by a service node failure. Note that, in this work, we do not consider the failure of compute nodes since a compute node usually supports a limited number of VMs, eg., 10.8 VM instances on a compute node [21], [22]. Thus, the failure of a compute node will not significantly degrade the performance from the cloud's perspective. Actually, the proposed algorithm in our work can be easily extended to deal with compute node failures (see details in Section IV-D). The main contributions of this paper are as follows:

1) We first discuss two abnormal situations, malicious tenants and service node failures. Accordingly, we propose two novel constraints, and the problem of VM placement that can alleviate the impact of abnormal events (VMP-AI) in the multi-tenant cloud.
2) We formulate the VMP-AI problem as an integer linear programming and prove its NP-hardness. Moreover, we propose a randomized rounding-based algorithm named R-VMP-AI for this problem and further prove it can achieve the bi-criteria approximation.
3) We evaluate the performance of the proposed algorithm through large-scale simulations and small-scale experiments compared with state-of-the-art solutions. The simulation results show that under the premise of cloud load balancing, our algorithm reduces the impact scope of service node failures by 60%, the impact scope of malicious tenants by 40%, and the tenant task makespan by 25% compared to state-of-the-art solutions.

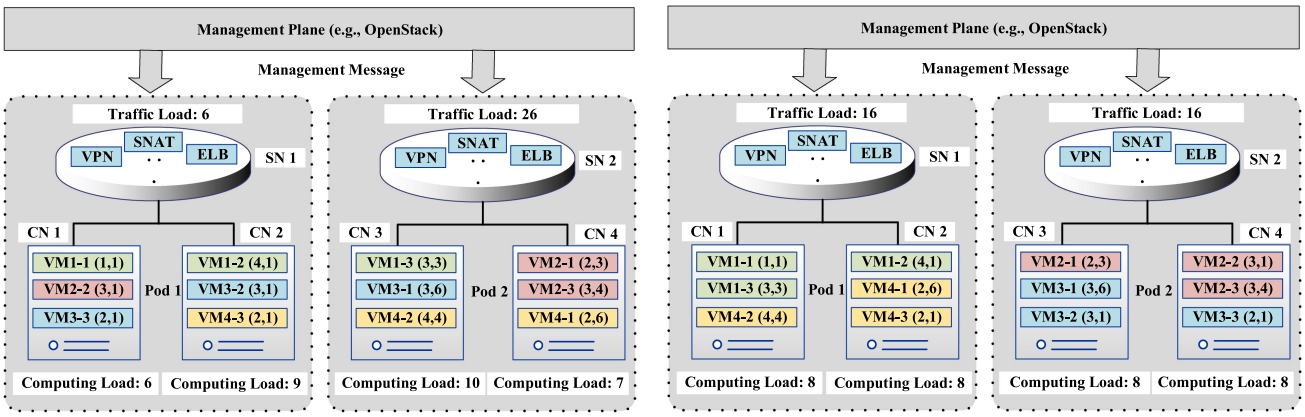
The rest of this paper is organized as follows. Section II presents the related works, including VM placement, malicious tenants, and service node failures. Section III introduces the motivation, describes the system model, defines the VMP-AI problem, and analyzes its complexity. In Section IV, we propose an efficient algorithm to solve VMP-AI, and theoretically analyze the approximate performance. Section V shows the simulation and experiment results of our proposed algorithm and some alternatives. We conclude this paper in Section VI.

II. Related Works

To provide stable cloud services, the cloud platform expects to minimize the negative impact of abnormal events while meeting the resource requirements (ie., computing resources and network resources). In this section, we summarize the state-of-the-art VM placement solutions and the methods of handling abnormal events in the cloud.

In recent years, a series of VM placement mechanisms [24], [25], [26], [27] have been widely proposed for improving the utilization of cloud resources and increasing cloud revenue. For example, to reduce the number of used cloud servers, a general way is to treat each compute node as a bin with fixed computing resources, each VM as a package, and the VM placement problem is formalized into a multi-dimensional bin-packing problem [24]. To improve the utilization of computing resources and network resources simultaneously, a multipath routing capability and dynamic VM migration algorithm is proposed [25]. Moreover, COmpvm [27] considered the VM placement based on the life cycle and resource requirements of VMs. However, the above VM placement mechanisms did not consider the scenario when abnormal events occur in the cloud.

Abnormal events in multi-tenant clouds are caused by external or internal factors, and attacks by malicious tenants and service node failures are common abnormal events caused by external and internal factors, respectively [6], [7]. On the one hand, malicious tenants are a huge threat because they will launch attacks [28] and make the service node unavailable [29] from inside, and it is difficult to be identified immediately. To prevent malicious tenants, a series of network isolation mechanisms and detection mechanisms have been proposed [11], [20], [30], [31]. The general solution is based on traffic analysis and packet capture. For example, seawall [20] deployed a traffic analysis module on the hypervisor for detecting the UDP traffic or abnormally behaving TCP stack from the malicious tenants. Once malicious traffic is detected, the security module may limit the malicious traffic speed or shut down the malicious VM [20]. Similarly, the packet captured from the network router can also detect whether there exists an attack [30]. However, the continuous traffic analysis and packet capture at scale come with prohibitive performance overheads. To reduce the overhead of traffic analysis, Privateeye [11] detects the malicious VM based on the 10-minute flow pattern changes, but it still cannot achieve 100% malicious detection. In addition, to prevent co-location attacks by malicious tenants, a series of VM placement strategies have also been proposed [32], [33], [34], [35], [36]. For example, Ding et al. [33] design a VM placement strategy to reduce the hit rate of the malicious tenants and the loss rate of the targeted tenants to enhance cloud computing security. Liu et al. [34] measure the cost of security risks for a VM allocation by the estimated percentage of malicious users and



(a) *Left plot*: The VM placement by the Nova-CN scheme. (b) *Right plot*: The VM placement by our scheme.

Fig. 1. Illustration of a tiny cloud consisting of two pods. Each pod consists of one service node (SN) and two compute nodes (CNs). The computing resource capacity of each compute node is set as 10, and the traffic processing capacity of each service node is set as 30 for simplicity. VMs of four tenants are deployed on the compute nodes, and the service nodes process their traffic. VM1-1, VM1-2 and VM1-3 belong to tenant 1. VM2-1, VM2-2 and VM2-3 belong to tenant 2. VM3-1, VM3-2 and VM3-3 belong to tenant 3. VM4-1, VM4-2 and VM4-3 belong to tenant 4. The numeric tuples of a VM represent the requested computing resource and traffic resource of this VM. For example, tuple VM1-2 (4,1) denotes that VM1-2 will cost 4 units of computing resource and 1 unit of traffic resource. The two diagrams denote two different ways of VM placement scheme (the Nova-CN scheme [23] and our proposed scheme). (a) *Left plot:* The VM placement by the Nova-CN scheme. (b) *Right plot:* The VM placement by our scheme.

the number of servers and users in the cloud. Azar et al. [35] propose a randomization way of assigning VMs that makes the deployment of VMs unpredictable for the attacker. Li et al. [36] design a Vickrey-Clarke-Groves mechanism to migrate VMs periodically, so that malicious VMs cannot stay co-located with their target VM for a long time.

On the other hand, service node failures are also common, often occurring in some abnormal events (eg., connectivity errors, hardware faults, and resource overload [14], [37], [38]). These service node failures may lead to the service being unavailable and decrease the QoS of tenants [15], [37]. To deal with service node failures, several efficient solutions have been designed [39], [40], [41], [42], [43]. For example, Li et al. [39] deployed redundant service nodes to deal with node failures and maximize the network service revenue. Shang et al. [40] and Zhang et al. [41] deployed backup instances to minimize the backup cost under resource constraints. Inspired by machine learning, Li et al. [42] introduce an AIOps (Artificial Intelligence for IT Operations) solution for predicting node failures for an ultra-large-scale cloud computing platform. Caviglione et al. [43] propose a multi-objective approach using a heuristic algorithm based on a deep reinforcement learning framework to achieve a trade-off between the impact of hardware outages, energy consumption and quality of service.

Existing works mainly focus on the prevention and recovery of abnormal situations. Although detection mechanisms and backup services can reduce the frequency of abnormal events, it is unrealistic to eliminate the occurrence of abnormal events. Therefore, limiting the impact of abnormal events is essential. In this paper, we focus on how to limit the impact scope caused by malicious tenants and service node failures to alleviate negative impacts when abnormal events occur, which complements existing works.

III. Preliminaries

A. A Motivating Example

In this section, we provide a toy example to illustrate that an appropriate VM placement strategy can alleviate the negative impact scope on the cloud network facing with an abnormal event. In this example, the cloud consists of two pods: pod 1 and pod 2. Each pod contains one service node (SN) and two compute nodes (CNs), as shown in Fig. 1. For convenience, let $\mathcal{V}=\{\text{CN1},\text{CN2},\text{CN3},\text{CN4}\}$ and $\mathcal{S}=\{\text{SN1},\text{SN2}\}$ denote the set of compute nodes and service nodes, respectively. The traffic processing capacity $C(s)$ of each service node $s \in \mathcal{S}$ is set as 30, and the resource capacity $R(v)$ (eg., CPU) of each compute node $v \in \mathcal{V}$ is set as 10 for simplicity. There are four tenants, each of which requests three VMs. For example, tenant 1 requests three VMs, named VM1-1,1-2 and 1-3. The corresponding resource demand for compute nodes (eg., CPU) and traffic demand for service nodes are marked after the VMs.

As one of the most popular open-source platforms for private and public clouds, OpenStack [44] uses the nova-scheduler to decide how VMs should be placed among the compute nodes of the OpenStack cluster. Specifically, when the nova-scheduler [23], [45] receives a VM placement request, it first uses a filtering algorithm to get a list of candidate compute nodes (which compute nodes have resources to host the VM and meet VM requirements). Then, the nova-scheduler uses a weighting algorithm to rank the compute nodes from the filtered list and choose the most appropriate compute node. By default, the lower the load of compute node, the higher the ranking. If we use the default nova-scheduler method (named Nova-CN for convenience) to place these VMs on compute nodes, the final placement result is shown in the left plot of Fig. 1. Specifically, VMs 1-1, 2-2 and 3-3 are placed on CN1; VMs 1-2, 3-2 and 4-3

are placed on CN2; VMs 1-3, 3-1 and 4-2 are placed on CN3; and VMs 2-1, 2-3 and 4-1 are placed on CN4.

However, this placement scheme will lead to poor network performance when encountering malicious tenants or service node failures. First, we observe that each tenant will access 2 pods. That is, if there is a malicious tenant, malicious traffic (eg., DoS attacks) will be injected into two service nodes. It will degrade the whole network performance and affect the QoS of all tenants. Second, each service node serves the VM instances of four tenants. That is, if one of the service nodes fails, all tenants will be affected. Moreover, we measure the traffic load $\ell(s)$ of each service node $s \in \mathcal{S}$ and the computing load $\ell(v)$ of each compute node $v \in \mathcal{V}$, then find the maximum computing load of the compute nodes is 10, and the maximum traffic load of the service nodes is 26. Therefore, the load ratio of service nodes is $\lambda_{SN} = \max\{\ell(s)/R(s), s \in \mathcal{S}\} = 26/30 = 0.87$, and the load ratio of compute nodes is $\lambda_{CN} = \max\{\ell(v)/R(v), v \in \mathcal{V}\} = 10/10 = 1$.

B. Our Intuition

A question immediately following the above discussion is: *can we design a VM placement scheme for better alleviating the impact of abnormal events in clouds?* Specifically, this VM placement scheme should be able to avoid the above three disadvantages of the Nova-CN method. In other words, we expect to design a VM placement scheme that limits the impact scope of malicious tenants and service node failures in clouds, and simultaneously achieves load balancing on both compute nodes and service nodes. In this example, a feasible placement scheme is shown in the right plot of Fig. 1. Specifically, VMs 1-1, 1-3 and 4-2 are placed on CN1; VMs 1-2, 4-1 and 4-3 are placed on CN2; VMs 2-1, 3-1 and 3-2 are placed on CN3; and VMs 2-2, 2-3 and 3-3 are placed on CN4. Based on the above VM placement scheme, the negative impact of malicious tenants and service node failures can be alleviated. For example, a malicious tenant can only attack one service node since each tenant can only access 1 pod, while a service node failure will affect two tenants because each service node serves the VM instances of two tenants. Thus, this VM placement scheme can better alleviate the impact of abnormal events compared to the Nova-CN method. In addition, we find that the maximum load of the service and compute nodes are 16 and 8, respectively. The load ratio of service nodes is $\lambda_{SN} = \max\{\ell(s)/R(s), s \in \mathcal{S}\} = 16/30 = 0.53$, and the load ratio of compute nodes is $\lambda_{CN} = \max\{\ell(v)/R(v), v \in \mathcal{V}\} = 8/10 = 0.8$. Note that our schedule can achieve better load balancing compared with Nova-CN due to its smaller load ratio. The performance comparison between Nova-CN and our proposed scheme is summarized in Table I.

In this paper, *our goal is to design a VM placement algorithm in clouds to alleviate the impact of abnormal events better.* To this end, we introduce two novel constraints in the traditional VM placement problem: service node and tenant constraints. On the one hand, compared with existing methods, our proposed algorithm can effectively alleviate the negative impact scope when an abnormal event occurs through multi-constrained VM placement without consuming additional resources. On the other hand, since we limit the number of pods (h) each tenant can access and the number of tenants hosted by each pod (w) when deploying VMs, our proposed method may lead to lower resource utilization in the cloud compared with existing methods. However, when the cloud vendor sets the appropriate values of h and w (see details in Section V), our proposed method only causes a slight waste of resources. In fact, we can achieve the trade-off between resource utilization and the impact range of abnormal events by adjusting the value of h and w.

TABLE I
PERFORMANCE COMPARISON BETWEEN NOVA-CN AND OUR PROPOSED SCHEME, INCLUDING MAX. COMPUTING LOAD, LOAD RATIO OF COMPUTE NODES, MAX. TRAFFIC LOAD, LOAD RATIO OF SERVICE NODES, AVG. NUMBER OF PODS ACCESSED BY TENANTS AND AVG. NUMBER OF TENANTS SERVED BY SERVICE NODES

Schemes	Nova-CN	OURS
Max. Computing Load	10	8
Load Ratio of Compute Nodes	1	0.8
Max. Traffic Load	26	16
Load Ratio of Service Nodes	0.87	0.53
Avg. No. pods accessed by tenants	2	1
Avg. No. tenants served by service nodes	4	2

TABLE II
KEY NOTATIONS

Parameters	Description
$\mathcal{U}$	a set of tenants
$\mathcal{V}$	a set of compute nodes (CNs)
$\mathcal{S}$	a set of service nodes (SNs)
P_u	a set of VM instances requested by tenant u
u	a tenant in $\mathcal{U}$
v	a CN in $\mathcal{V}$
s	a SN in $\mathcal{S}$
$s(v)$	the SN that provides service for CN v
$p_{u,d}$	the dth VM instance requested by tenant u
$r(p_{u,d})$	the computing resource demand of VM $p_{u,d}$
$f(p_{u,d})$	the traffic demand of VM $f_{u,d}$
$R(v)$	the computing resource capacity on CN v
$b(v)$	the used computing resource on CN v
$C(s)$	the traffic processing capacity of SN s
$b(s)$	the used traffic resource of SN s
h	the maximum No. pods one tenant can access
w	the maximum No. tenants one pod can serve
$g(u,s)$	whether tenant u has requested resources in node s

Variables	Description
$x_v^{p_{u,d}}$	whether VM instance $p_{u,d}$ of tenant u is placed in CN v or not
y_s^u	whether SN s will process the traffic of tenant u or not
λ	the load balancing factor

C. System Model

In this section, we abstract the cloud into two components: the infrastructure model and the multi-tenant model. For ease of reference, Table II summarizes the key notations.

1) Infrastructure Model: A typical cloud consists of many pods, each of which usually consists of a set of compute nodes and one service node [46]. Compute nodes provide computing resources for tenants in the form of VMs. We use

$\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ to denote the set of compute nodes, where n is the number of compute nodes in the cloud. Service nodes provide various network services (eg., ELB [47], firewall [48]) for tenants. Let $\mathcal{S} = \{s_1, s_2, \ldots, s_q\}$ denote the service node set, where q is the number of pods in the cloud.

In addition, we use $s(v)$ to represent the service node that provides service for compute node $v \in \mathcal{V}$. In other words, service node $s(v)$ is responsible for providing services for VMs in compute node v. For each service node s, we use $C(s)$ to denote its traffic processing capacity. Since some tenants may have deployed VMs and generated traffic before, we use $b(s)$ to denote the existing background traffic in service node s. Moreover, let a binary constant $g(u, s)$ denote whether tenant u has requested traffic resources in service node s or not. For each compute node v, we use $R(v)$ to denote the computing resource (eg., CPU) capacity, and $b(v)$ to denote the amount of occupied resources. Note that, $R(v)$ and $b(v)$ can be expanded into resource vectors that represent different types of resources on compute node v.

2) Multi-Tenant Model: In a multi-tenant cloud, a set of tenants rent VMs and buy services from cloud vendors according to their demands. Let $\mathcal{U} = \{u_1, u_2, \ldots u_m\}$ denote the set of tenants, where m is the number of tenants in the cloud. Tenants usually request a set of VMs with different application requirements, such as batch processing and high-performance computing [49]. Therefore, the required resources of each VM are different. For each tenant $u \in \mathcal{U}$, let $\mathcal{P}_u = \{p_{u,1}, p_{u,2}, \ldots, p_{u,l_u}\}$ denote the set of requested VM instances, where l_u is the number of VM instances required by tenant u. Each VM instance $p_{u,d} \in \mathcal{P}_u$ with $1 \leq d \leq l_u$, will consume some compute resources (eg., CPU and RAM), denoted as $r(p_{u,d})$. Similar to $R(v)$ and $b(v)$, $r(p_{u,d})$ also can be expanded into a resource vector that represents different resource requirements. Moreover, for each VM instance $p_{u,d} \in \mathcal{P}_u$, we use $f(p_{u,d})$ to denote the traffic demand that needs to be served by the corresponding service node. In practice, $r(p_{u,d})$ is specified when the VM instance $p_{u,d}$ is created, and $f(p_{u,d})$ can be estimated according to the fees paid by tenant u and the type of VM instance $p_{u,d}$ [50].

D. Problem Definition

We formally define the VM placement problem that can alleviate the impact of abnormal events (VMP-AI) with the following three constraints: 1) *Sevice node constraints:* To alleviate the impact of the service node failure, we limit the number of tenants served by each service node, so that the number of affected tenants by a service node failure is controllable. Specifically, each service node can serve h tenants at most. 2) *Tenant constraints:* To alleviate the impact of the malicious tenants, we limit the number of service nodes/pods that each tenant can access, so that the number of affected service nodes/pods is controllable when encountering malicious tenants attacks. Specifically, we should ensure that the traffic of each tenant can be forwarded to w service nodes at most. 3) *Resource constraints:* The resources that each compute node can provide must meet the needs of the tenants on that compute node. Specifically, for each compute node, its resource load should not exceed $\lambda \cdot R(v)$. For each service node, its traffic load should not exceed $\lambda \cdot C(s)$. The goal of VMP-AI is to achieve the load balance among all service nodes and all compute nodes. Accordingly, we formulate the VMP-AI problem as follows:

$$\min \lambda$$

$$S.t. \begin{cases} \sum_{v \in \mathcal{V}} x_v^{p_{u,d}} = 1, & \forall u, d \\ x_v^{u,d} \leq y_{s(v)}^u, & \forall u, v, d \\ g(u,s) \leq y_s^u, & \forall u, s \\ \sum_{u \in \mathcal{U}} y_s^u \leq h & \forall s \\ \sum_{s \in \mathcal{S}} y_s^u \leq w & \forall u \\ \sum_{u \in \mathcal{U}} \sum_{p_{u,d} \in \mathcal{P}_u} \sum_{v \in \mathcal{V}: s(v)=s} x_v^{p_{u,d}} \cdot f(p_{u,d}) & \\ \quad + b(s) \leq C(s) \cdot \lambda & \forall s \\ \sum_{u \in \mathcal{U}} \sum_{p_{u,d} \in \mathcal{P}_u} x_v^{p_{u,d}} \cdot r(p_{u,d}) & \\ \quad + b(v) \leq R(v) \cdot \lambda, & \forall v \\ x_v^{p_{u,d}} \in \{0,1\}, & \forall u, v, d \\ y_s^u \in \{0,1\}, & \forall u, s \end{cases} \tag{1}$$

In (1), $x_v^{p_{u,d}}$ denotes whether VM instance $p_{u,d}$ of tenant u is placed in compute node v or not. $y_s^u \in \{0,1\}$ represents whether service node s will process the traffic of tenant u or not. The first set of equations means that each VM will be placed on one and only one compute node. The second and third sets of inequalities express that once a VM of tenant u is placed on compute node v, the connected service node $s(v)$ will process the traffic from tenant u (ie., $y_{s(v)}^u = 1$). Similarly, if service node s has background traffic from tenant u (ie., $g(u,s) = 1$), we have $y_s^u = 1$. The fourth set of inequalities represents the Sevice node constraints, ie., the number of tenants that a service node can serve cannot exceed h. The fifth set of inequalities means the Tenant constraints, ie., the number of service nodes accessed by each tenant cannot exceed w. The sixth and seventh sets of inequalities indicate the resource load on each compute node v_j and the traffic load of each service node, where λ is the load balancing factor. Our goal is to achieve the load balancing among all service nodes and among all compute nodes, ie., $\min \lambda$.

E. Problem Complexity Analysis

In this section, we analyze the complexity of the VMP-AI problem. Specifically, we consider the following six constraints in VMP-AI. (1) Each requested VM needs to be placed on one compute node. (2) The service traffic of each VM will be processed by the corresponding service node. (3) The number of tenants served by each service node is constrained. (4) The number of service nodes accessed by each tenant is constrained. (5) Each compute node should not be overloaded. (6) Each service node should not be overloaded. In fact, even if some constraints are relaxed, the problem is still challenging, which illustrates the complexity of our problem. By ignoring some constraints, we can transform VMP-AI into two classic NP-hard problems: the unrelated parallel machine scheduling (UPMS) problem [51] and the multiple commodity flow (MCF) problem [52]. As a result, the VMP-AI problem is NP-hard too.

Unrelated Parallel Machine Scheduing (UPMS) problem [51]: There exists m parallel machines and n independent

jobs. Each job is assigned to one of the machines. The processing of job j on machine i requires time $p_{i,j}$. The objective is to find a schedule that minimizes the makespan. According to [51], there exist a complex algorithm based on combinatorial rounding with the approximate factor of 2.

Difference from the UPMS problem: If we ignore the constraints (3), (4), (5), (6), and the goal is to achieve the load balancing among all compute nodes. We regard each compute node as a machine, and each VM as an independent job. Thus, the VMP-AI problem becomes the UPMS problem, ie., the UPMS problem is a special case of VMP-AI.

Multiple Commodity Flow (MCF) problem [52]: There exists k different commodities in a network with known topology. The network consists of n nodes and m edges. Each commodity is composed of a source node s, a destination node t, and a demand vector $\vec{d} = (d_1, d_2, \ldots)$, denoted as $(s, t, \vec{d})$. The objective of this problem is to find the percentage of load balancing that can simultaneously transport a set of commodities without violating the capacity constraints. According to [53], there exists an efficient algorithm with the approximate factor of $1+\epsilon$, and its time complexity is $O(k^{2.5}a^2b^{2.5}\log(a\epsilon^{-1})DU)$, where k is the number of commodities, a and b denote the number of nodes and edges in the network, D is the greatest demand, and U is the largest edge capacity.

Difference from the MCF problem: We use R and C to denote the maximum computing capacity of compute nodes and the maximum processing capacity of service nodes, respectively. Then we construct a network topology with a source node s, a destination node t, and $m+n$ internal nodes. We name these internal nodes as $\{v_1, \ldots, v_m\}$ and $\{s_1, \ldots, s_n\}$. The source node s connects with each internal node v_i with link capacity $(R(v_i), C)$, and the destination node t connects with each internal node s_j with link capacity $(R, C(s_j))$. Moreover, if the traffic from compute node v_i will be processed by service node s_j, then we construct a link between node v_i and node s_j with link capacity (R, C). Then we ignore constraints (3) and (4) of VMP-AI, and regard each VM as a commodity with $(s, t, \vec{d})$, where $\vec{d} = (d_1, d_2)$. Here d_1 and d_2 represent the requested computing resource and traffic resource of this VM, respectively. The goal of VMP-AI is to achieve load balancing. Thus, the VMP-AI problem becomes the MCF problem. Therefore, the MCF problem is a special case of VMP-AI.

Based on the above analyses, designing an algorithm with bounded approximation factors for VMP-AI is far from trivial and in urgent need.

IV. Algorithm Design

In this section, we present a rounding-based algorithm to solve the VMP-AI problem, called R-VMP-AI, and analyze the approximate performance.

A. Algorithm Description

The R-VMP-AI algorithm starts by constructing the linear programming as relaxation of VMP-AI (LP-VMP-AI). More specifically, LP-VMP-AI assumes that both the VM placement and the VM's traffic demands are splittable. That is, LP-VMP-AI relaxes the variables $\{x_v^{p_{u,d}}\}$ and $\{y_s^u\}$ from integral to fractional. Since LP-VMP-AI is linear programming, we can solve it with a linear programming solver in polynomial times, and get the optimal solution $\{\widetilde{x}_v^{p_{u,d}}\}$ and $\{\widetilde{y}_s^u\}$ (lines 2-4), and the optimal result is denoted as $\widetilde{\lambda}$. As LP-VMP-AI is relaxation of VMP-AI, $\widetilde{\lambda}$ is the lower-bound result for VMP-AI. The second step is to derive an integer solution $\{\widehat{y}_s^u\}$, for $\forall s \in \mathcal{S}$ and $\forall u \in \mathcal{U}$, by randomized rounding [54]. For each individual tenant u and service node s, R-VMP-AI rounds variable $\widehat{y}_s^u$ to 1 with probability $\widetilde{y}_s^u$ (lines 5-13) to keep the service node constraints and the tenant constraints. If $\widehat{y}_s^u = 1$, it means that the traffic of tenant u can be served by the service node s. Each rounding decision is independent with each other. Then the third step is to decide the VM placement scheme by rounding variables $\widehat{x}_v^{p_{u,d}}$ to 1 with probability $\widetilde{x}_v^{p_{u,d}}/\widetilde{y}_s^u$. If $\widehat{x}_v^{p_{u,d}} = 1$, then we place VM $p_{u,d}$ on compute node v (lines 14-26). Based on the above process, we get an integer solution $\{\widehat{x}_v^{p_{u,d}}, \widehat{y}_s^u\}$. R-VMP-AI is formally described in Algorithm 1.

Algorithm 1: R-VMP-AI: Rounding-Based Algorithm for VMP-AI.

1: Input: System parameters $\{h, w\}$, resource parameters $\{R(v), C(s), b(v), b(s)\}$, and tenant VM requests $\{P_u\}$
2: **Step 1: Sloving the relaxation problem of VMP-AI**
3: Construct a linear programming named LP-VMP-AI formalized in (1)
4: Obtain the optimal fractional solutions $\{\widetilde{x}_v^{p_{u,d}}, \widetilde{y}_s^u\}$
5: **Step 2: Selecting service nodes for tenants**
6: **for** *each tenant* $u \in \mathcal{U}$ **do**
7: **for** *each service node* $s \in \mathcal{S}$ **do**
8: *Set* $\widehat{y}_s^u = 1$ *with probability* $\widetilde{y}_s^u$
9: **if** $\widehat{y}_s^u == 1$ **then**
10: *Select service node* s *for tenant* u
11: **Step 3: Placing VM instances on compute nodes**
12: **for** *each tenant* $u \in \mathcal{U}$ **do**
13: **for** *each requested VM* $p_{u,d} \in \mathcal{P}_u$ **do**
14: **for** *each compute node* $v \in \mathcal{V}$ **do**
15: **if** $\widehat{y}_{s(v)}^u == 1$ **then**
16: *Set* $\widehat{x}_v^{p_{u,d}} = 1$ *with probability* $\frac{\widetilde{x}_v^{p_{u,d}}}{\widetilde{y}_{s(v)}^u}$
17: **if** $\widehat{x}_v^{p_{u,d}} == 1$ **then**
18: *Place VM* $p_{u,d}$ *on compute node* v
19: *Output: VM placement schemes* $\{\widehat{x}_v^{p_{u,d}}, \widehat{y}_s^u\}$

B. Performance Analysis

First, we prove that the solution of the algorithm satisfies the constraints in (1) on expectations. Then we analyze the approximate performance of R-VMP-AI.

Lemma 1. The R-VMP-AI algorithm allocates sufficient resources for these requested VMs with a high probability.

Proof. First, by construction, R-VMP-AI first selects a set of service nodes for tenants (lines 5-13), then places VMs on the compute nodes connected to the service nodes belong to this node set (lines 14-26), which satisfies the third set of constraints

in (1). Then we have the probability that VM $p_{u,d}$ is placed on compute node v:

$$\begin{aligned} Pr\left[\widehat{x}_v^{p_{u,d}}=1\right] &= Pr\left[\widehat{x}_v^{p_{u,d}}=1|\widehat{y}^u_{s(v)}=1\right] Pr\left[\widehat{y}^u_{s(v)}=1\right] \\ &\quad + Pr\left[\widehat{x}_v^{p_{u,d}}=1|\widehat{y}^u_{s(v)}=0\right] Pr\left[\widehat{y}^u_{s(v)}=0\right] \\ &= \frac{\widetilde{x}_v^{p_{u,d}}}{\widetilde{y}^u_{s(v)}} \cdot \widetilde{y}^u_{s(v)} = \widetilde{x}_v^{p_{u,d}} \end{aligned} \tag{2}$$

On the other hand, according to the first set of constraints in (1), the sum of the probabilities that VM instance $p_{u,d}$ being placed on each compute node is:

$$\sum_{v\in\mathcal{V}} Pr\left[\widehat{x}_v^{p_{u,d}}=1\right] = \sum_{v\in\mathcal{V}} \widetilde{x}_v^{p_{u,d}} = 1 \tag{3}$$

The above analysis shows that the compute nodes provide enough CPU and RAM resources for VM instances with a high probability.

Lemma 2. The solution returned by R-VMP-AI satisfies the compute node resource constraint and service node traffic constraint.

Proof. According to the the second set of constraints in (1), we have the resources expected to be allocated to all tenants in compute node v:

$$\begin{aligned} &\mathbb{E}\left[\sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in\mathcal{P}_u} \widehat{x}_v^{p_{u,d}} \cdot r(p_{u,d})\right] + b(v) \\ &= \sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in\mathcal{P}_u} Pr\left[\widehat{x}_v^{p_{u,d}}=1\right] \cdot r(p_{u,d}) + b(v) \\ &= \sum_{u_i\in\mathcal{U}}\sum_{p_{u,d}\in\mathcal{P}_u} \widetilde{x}_v^{p_{u,d}} \cdot r(p_{u,d}) + b(v) \le R(v)\cdot\widetilde{\lambda} \end{aligned} \tag{4}$$

The second equation is because the variable $\widehat{x}_v^{p_{u,d}}$ equals 1 with probability $\widetilde{x}_v^{p_{u,d}}$. Similarly, we can prove the service node traffic constraint is also satisfied. Thus, we omit the proof here.

Lemma 3. The solution of R-VMP-AI also satisfies the service node constraints and the tenant constraints in (1).

Proof. We prove the service node constraints as an example. According to the the six set of constraints in (1), we have the expected number of tenants to be served by service node s.

$$\mathbb{E}\left[\sum_{u\in\mathcal{U}} \widehat{y}^u_s\right] = \sum_{u\in\mathcal{U}} \Pr\left[\widehat{y}^u_s=1\right] = \sum_{u\in\mathcal{U}} \widetilde{y}^u_s \le h \tag{5}$$

The second equation is because the variable $\widehat{y}^u_s$ equals to 1 with probability $\widetilde{y}^u_s$.

The above analysis has shown that R-VMP-AI satisfies in expectation all the constraints in (1). However, these constraints may be violated. Next, we analyze the approximate performance of R-VMP-AI. In the approximate performance analysis, we use two classical theorems, Chernoff Bound and Union Bound. First, we define a variable α to assist our proof as follows:

$$\begin{aligned} \alpha = \min\left\{\frac{\widetilde{\lambda}\cdot R(v)-b(v)}{r(p_{u,d})}, \frac{\widetilde{\lambda}\cdot C(s)-b(s)}{f(p_{u,d})}, h, w\right\}, \\ v\in V, u\in\mathcal{U}, s\in\mathcal{S} \end{aligned} \tag{6}$$

On the right side of (6), the first item denotes the ratio of the available resource of compute node to the computing resource demand of each VM; the second item represents the ratio of the available capacity of service node to the traffic demand of each VM; the third item and the last item denote the number of tenants served by one service node and the number of service nodes one tenant can access, respectively. Next, we give the approximate performance analysis in detail.

Theorem 4. The R-VMP-AI algorithm can achieve the approximation factor of $\frac{2\log n}{\alpha}+3$ for the resource constraint of compute nodes, where n is the number of compute nodes.

Proof. Before analysing the resource constraint of compute nodes, we define a variable $\sigma_v^{p_{u,d}}$ to denote the occupied resource for each VM $p_{u,d}$ on each compute node v:

$$\sigma_v^{p_{u,d}} = \begin{cases} r(p_{u,d}), & with\ probability\ of\ \ \widetilde{x}_v^{p_{u,d}} \\ 0, & otherwise. \end{cases} \tag{7}$$

According to the definition, $\{\sigma_v^{p_{u,d}}\}$ are mutually independent. The resource expected to be occupied on compute node v is as follows:

$$\begin{aligned} &\mathbb{E}\left[\sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in P_u} \sigma_v^{p_{u,d}}\right] + b(v) \\ &= \sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in P_u} \mathbb{E}[\sigma_v^{p_{u,d}}] + b(v) \\ &= \sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in P_u} \widetilde{x}_v^{p_{u,d}}\cdot r(p_{u,d}) + b(v) \le R(v)\cdot\widetilde{\lambda} \end{aligned} \tag{8}$$

Combining (8) and the definition of α in (6), we have:

$$\begin{cases} \frac{\sigma_v^{p_{u,d}}\cdot\alpha}{R(v)\cdot\widetilde{\lambda}-b(v)} \in [0,1] \\ \mathbb{E}\left[\sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in P_u} \frac{\sigma_v^{p_{u,d}}\cdot\alpha}{R(v)\cdot\widetilde{\lambda}-b(v)}\right] \le \alpha. \end{cases} \tag{9}$$

Then by applying Chernoff Bound, assume that ρ is an arbitrary positive value. It follows:

$$\begin{aligned} &\mathbf{Pr}\left[\sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in P_u} \frac{\sigma_v^{p_{u,d}}\cdot\alpha}{R(v)\cdot\widetilde{\lambda}-b(v)} \ge (1+\rho)\cdot\alpha\right] \le e^{\frac{-\rho^2\cdot\alpha}{2+\rho}} \\ \Rightarrow &\mathbf{Pr}\left[\sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in P_u} \frac{\sigma_v^{p_{u,d}}}{R(v)\cdot\widetilde{\lambda}-b(v)} \ge (1+\rho)\right] \le e^{\frac{-\rho^2\cdot\alpha}{2+\rho}} \le \frac{1}{n^2} \end{aligned} \tag{10}$$

Where n denotes the number of compute nodes, and $\frac{1}{n^2}$ is a value close to zero. By solving the above equation, we have:

$$\rho \ge \frac{2\log n}{\alpha} + 2 \tag{11}$$

By applying Union Bound, we have the probability that any server violates the constraint equals to:

$$\begin{aligned}&\mathbf{Pr}\left[\bigcup_{v\in\mathcal{V}}\left\{\sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in P_u}\frac{\sigma_v^{p_{u,d}}\cdot\alpha}{R(v)\cdot\widetilde{\lambda}-b(v)}\geq(1+\rho)\cdot\alpha\right\}\right]\\&\leq\sum_{v\in\mathcal{V}}\mathbf{Pr}\left[\sum_{u\in\mathcal{U}}\sum_{p_{u,d}\in P_u}\frac{\sigma_v^{p_{u,d}}\cdot\alpha}{R(v)\cdot\widetilde{\lambda}-b(v)}\geq(1+\rho)\cdot\alpha\right]\\&\leq n\cdot\frac{1}{n^2}=\frac{1}{n},\rho\geq\frac{2\log n}{\alpha}+2\end{aligned}\tag{12}$$

Note that the second equality holds, because the number of compute nodes equals n. The resource violation of our algorithm will not exceed $\rho+1=\frac{2\log n}{\alpha}+3$.

Theorem 5. The R-VMP-AI algorithm can achieve the approximation factor of $\frac{2\log q}{\alpha}+3$ for the service node traffic constraint, where q denotes the number of service nodes.

Proof. We can prove Theorem 5 in a similar way with the proof of Theorem 4. Thus, we omit the detailed proof here.

Theorem 6. The R-VMP-AI algorithm can achieve the approximation factor of $\frac{2\log q}{\alpha}+3$ for the service node constraints.

Proof. We define a random variable θ_s^u to denote whether service node s serves the traffic from tenant u:

$$\theta_s^u=\begin{cases}1, & with\,probability\,of\quad\widetilde{y}_u^s\\0, & otherwise.\end{cases}\tag{13}$$

According to the definition, $\{\theta_s^u\}$ are mutually independent. The expected number of tenants served on service node s is:

$$\mathbb{E}\left[\sum_{u\in\mathcal{U}}\theta_s^u\right]=\sum_{u\in\mathcal{U}}\mathbb{E}[\theta_s^u]=\sum_{u\in\mathcal{U}}\widetilde{y}_s^u\leq h\tag{14}$$

Combining (14) and the definition of α in Def. (6), we have:

$$\begin{cases}\frac{\theta_s^u\cdot\alpha}{h}\in[0,1]\\\mathbb{E}\left[\sum_{u\in\mathcal{U}}\frac{\theta_s^u\cdot\alpha}{h}\right]\leq\alpha.\end{cases}\tag{15}$$

Then by applying Chernoff Bound, assume that τ is an arbitrary positive value. It follows:

$$\begin{aligned}&\mathbf{Pr}\left[\sum_{u\in\mathcal{U}}\frac{\theta_s^u\cdot\alpha}{\Omega}\geq(1+\tau)\cdot\alpha\right]\leq e^{\frac{-\tau^2\cdot\alpha}{2+\tau}}\\&\Rightarrow\mathbf{Pr}\left[\sum_{u\in\mathcal{U}}\frac{\theta_s^u}{\Omega}\geq(1+\tau)\right]\leq e^{\frac{-\tau^2\cdot\alpha}{2+\tau}}\leq\frac{1}{q^2}\end{aligned}\tag{16}$$

Where q denotes the number of service nodes, and $\frac{1}{q^2}$ is a value close to zero. By solving the above equation, we have:

$$\tau\geq\frac{2\log q}{\alpha}+2\tag{17}$$

By applying Union Bound, we have the probability that any server violates the constraint equals to:

$$\begin{aligned}&\mathbf{Pr}\left[\bigcup_{s\in\mathcal{S}}\left\{\sum_{u\in\mathcal{U}}\frac{\theta_s^u\cdot\alpha}{\Omega}\geq(1+\tau)\cdot\alpha\right\}\right]\\&\leq\sum_{s\in\mathcal{S}}\mathbf{Pr}\left[\sum_{u\in\mathcal{U}}\frac{\theta_s^u\cdot\alpha}{\Omega}\geq(1+\tau)\cdot\alpha\right]\\&\leq q\cdot\frac{1}{q^2}=\frac{1}{q},\rho\geq\frac{2\log q}{\alpha}+2\end{aligned}\tag{18}$$

The second equality holds because the number of service nodes equals q. The resource violation of our algorithm will not exceed $\tau+1=\frac{2\log q}{\alpha}+3$.

Theorem 7. The R-VMP-AI algorithm can achieve the approximation factor of $\frac{2\log q}{\alpha}+3$ for the tenant constraints.

Proof. We can prove Theorem 7 in a similar way to the proof of Theorem 6. Thus we omit the detailed proof here.

Approximation Factor: According to the above analysis, we can conclude that the approximate factors of our algorithm are *bi-criteria approximations* with respect to both the objective value and constraints. In many practical scenarios, these factors are constant. For example, consider a cloud with thousands of compute nodes and hundreds of service nodes. Then we have $n=1000$ and $q=100$. In general, one compute node can accommodate more than ten VMs, and there are more than ten compute nodes in one pod. Thus, we estimate the value $\alpha=10$. Then the bi-criteria approximation factor becomes $\frac{2\log n}{\alpha}+3=4.38$ and $\frac{2\log q}{\alpha}+3=3.92$, respectively.

C. Complete Algorithm Description

In fact, the output of the R-VMP-AI algorithm may violate some constraints in (1). For example, one VM instance may be placed on multiple compute nodes, or it may not be placed. Moreover, the VM placement scheme may violate the resource constraints of compute nodes and service nodes. To deal with such cases, the VM placement needs to convert the bi-criteria solution into a feasible solution, ie., a solution that satisfies the constraints in (1), thereby making the algorithm more practical. To obtain the feasible VM placement solution, we mainly consider the following four steps. The first step is the same as that of R-VMP-AI: construct linear programming by relaxing the binary variables in (1) and obtain the optimal fractional solution $\{\widetilde{x}_v^{p_{u,d}},\widetilde{y}_s^u\}$.

In the second step, we first transform the fractional solution $\{\widetilde{y}_s^u\}$ to several feasible integral solutions $\{\widehat{y}_s^u\}$, and these integral solutions satisfy the constraints in (1). For example, for tenant u_1 and service node s_1, we have the optimal soution $\widetilde{y}_s^u=\{0.8,0.4,0\}$ and we have the constraint $\sum y_s^u\leq 2$. Then we can transform the fractional solution $\widetilde{y}_s^u$ to three feasible integral solutions $\widehat{y}_{s_1}^{u_1}=\{1,1,0\}$, $\{1,0,0\}$, and $\{0,1,0\}$.

The third step is to find the feasible integral solutions $\{\widehat{x}_v^{p_{u,d}}\}$ for the generated integral solution $\{\widehat{y}_s^u\}$, and these integrated solutions can strictly meet the computing resource constraint and the traffic resource constraint in (1). For example, once there exists a tenant u and service node s to satisfy $\widehat{y}_s^u=0$, then all compute nodes connected to the service node cannot place the VM of tenant u. Since $\{\widehat{y}_s^u\}$ strictly satisfy the constraints, we can use the greedy method to find several feasible integral solutions $\{\widehat{x}_v^{p_{u,d}}\}$. Finally, based on the above three steps, we will find some feasible VM placement schemes which satisfy

the constraints in (1). The last step is to compare these integral solutions, and choose the minimum λ as the output solution. Moreover, we can decrease the time complexity of the algorithm by limiting the number of feasible solutions $\{\widehat{y}_s^u\}$ and $\{\widehat{x}_v^{p_{u,d}}\}$ in the second and third steps, respectively.

D. Discussion

- In some scenarios, cloud vendors want to alleviate the impact of compute node failures on the cloud. We claim that our proposed algorithm also works in this case. Specifically, let $z_v^u \in \{0,1\}$ denote whether some VMs of tenant u will be placed on compute node v or not. According to the definitions of z_v^u and $x_v^{p_{u,d}}$, we have:

$$z_v^u \geq x_v^{p_{u,d}}, \quad \forall u, v, d \tag{19}$$

  Similar to service node constraints, we limit the number of tenants served by each compute node (ie., each compute node can serve q tenants at most), so that the number of affected tenants by a compute node failure is controllable. In this way, we have *compute node constraints*, as shown in (20).

$$\sum_{u \in \mathcal{U}} z_v^u \leq q, \quad \forall v \tag{20}$$

  The compute node constraints are similar to the service node constraints, so our proposed algorithm can be extended to address this scenario.

V. Performance Evaluation

A. Performance Metrics and Benchmarks

1) Performance Metrics: We use the following nine performance metrics to evaluate the performance of our proposed algorithm. (1) The load ratio of compute nodes (CNs); (2) The load ratio of service nodes (SNs); (3) The impact scope of service node failures (ie., the average number of tenants served by each service node); (4) The impact scope of malicious tenants (ie., the average number of service nodes that each tenant accesses); (5) The valid cloud throughput; (6) The CPU utilization of each service node; (7) The round-trip time (RTT) of each service node; (8) The packet loss ratio of each service node; and (9) The task makespan of each tenant.

In large-scale simulations, we use the first and second metrics to evaluate the load balancing among all compute nodes and service nodes, respectively. The third and fourth metrics are used to evaluate the scope of negative impact on the cloud network when abnormal events occur, ie., the impact scope by malicious tenants and the impact scope by service node failures. We use the largest resource load ratio of all compute nodes as the first metric. For each service node, its load ratio is its traffic load divided by its traffic processing capacity, and we use the largest value as the second metric. For each compute node, its resource load ratio is the maximum utilization of its CPU load ratio and RAM load ratio. The CPU/RAM load ratio equals the CPU/RAM load of these VMs divided by the CPU/RAM capacity of this compute node. Moreover, we measure the average number of tenants served by each service node and the average number of service nodes that tenants access as the third metric and the fourth metric, respectively. We compute the valid cloud throughput as the fifth metric. The valid cloud throughput is equal to the sum of traffic that satisfies the service node constraints and the tenant constraints.

In the small-scale experiments, we use iperf3 [55] and hping [56] to implement the normal traffic and malicious tenant traffic, respectively. Moreover, we measure the CPU utilization, the RTT, and the packet loss ratio of service nodes as the sixth to eighth metrics. Since tenants may generate some tasks, we collect the tenant task makespan as the ninth metric, which is the completion time of all tasks for the tenant.

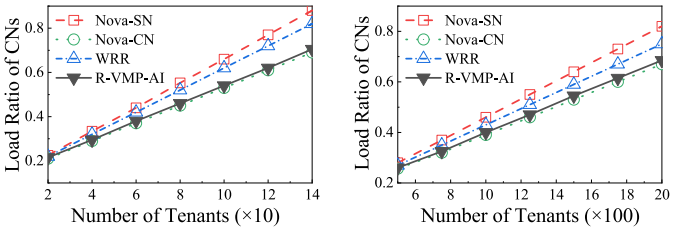
2) Benchmarks: We choose three benchmarks for performance comparison. The first benchmark is the default nova-schedule scheme [23], [45]. It first filters a list of candidate compute nodes, then places the VM on the compute node with the lightest load according to a weighting algorithm. For convenience, we named the default scheduling of the nova-scheduler as Nova-CN (see details in Section III-A). Since Nova-CN does not pay attention to the workload of service nodes, we reconfigure the weighting algorithm of the nova-scheduler and name it Nova-SN as the second benchmark. Specifically, after the filtering algorithm of the nova-scheduler, the weighting algorithm first chooses one service node with the least workload. Then it selects the idlest compute node from the compute node set served by this service node to place VMs. Nova-SN represents a category solution that separately considers the remaining traffic processing capacity of service nodes and the remaining computing resource of compute nodes. The last benchmark is the Weight-Round-Robin (WRR) [57] method. WRR first allocates weights to each service node based on its remaining resources and the remaining resources of the compute nodes served by it. Then WRR generates a random value and selects the service node according to the value. Finally, WRR selects a compute node from the compute node set served by the service node to place VMs.

B. Large-Scale Simulations

1) Simulation Settings: We perform our simulations over two infrastructure clouds. The first cloud architecture is Koala [59], which consists of 20 service nodes and 600 compute nodes. Each service node serves the traffic of 30 compute nodes. The second cloud is generated based on the Google cluster-data [60], which consists of 10047 compute nodes and 324 service nodes. Each service node serves the traffic of 25-35 compute nodes. As mentioned in Section III, one or more types of service nodes do not affect our simulation results. For simplicity, we only consider one type of service node. We generate three different types of VMs: standard, memory-optimized, and computing. The above three types of VMs are cloned in Tencent Clouds [58], and their required resources (vCPU, RAM, and bandwidth) are different. As shown in Table III, the first three instances represent the standard VM instances, accommodating most applications, such as streaming media businesses and online-game [58]. The next three instances represent the memory-optimized

TABLE III
MULTIPLE VM INSTANCES WITH DETAILED RESOURCES DEMAND FROM TENCENT CLOUD [58]

Name	vCPU	RAM(GB)	Bandwidth(Gbps)
s5.small2	1	2	1.5
s5.medium4	2	4	1.5
s5.large8	4	8	1.5
m5.small8	1	8	1.5
m5.medium16	2	16	1.5
m5.large32	4	32	1.5
c4.small2	1	2	3.0
c4.medium4	2	4	3.0
c4.large8	4	8	3.0

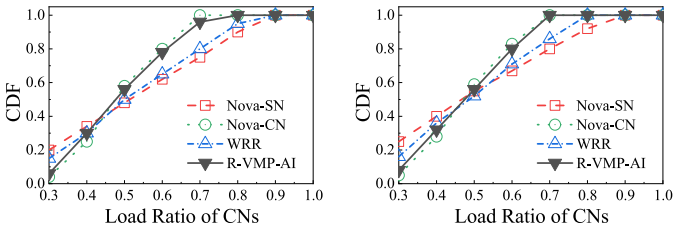


(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 2. Load ratio of compute nodes versus number of tenants.

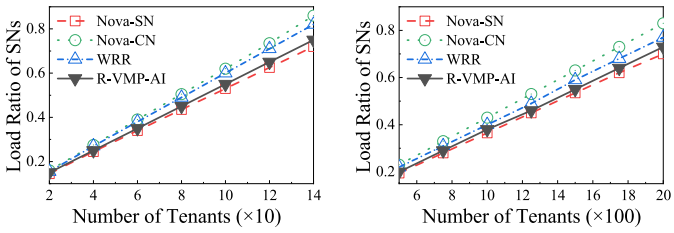
VM instances. They are suitable for applications that require extensive memory operations, searches, and computations, such as high-performance databases and distributed memory caching [58]. The last three instances are computing VM instances, which are suitable for compute-intensive workloads such as batch processing, high-performance computing, and dedicated game servers [58]. The number of tenants in the above two clouds is set as 140 and 2000, respectively. Each tenant randomly creates 1-50 VM instances from Table III with different types. For each compute node, we set its vCPU cores and RAM capacity as 25 and 150 GB, respectively. The traffic processing capacity of each service node is set as 500 Gbps, and the system parameters h and w are set as 40 and 10 by default, respectively. We run each simulation 30 times and calculate the average value as the simulation results.

2) Simulation Results: The first set of simulations compares the load balancing performance (ie., the compute node load ratio and the service node load ratio) of R-VMP-AI with three benchmarks. In Fig. 2, as the number of tenants increases, the load ratio of compute nodes accordingly increases in both the Koala-based cloud and the Google-based cloud. For the load ratio of compute nodes, the increasing rate of R-VMP-AI is much slower than that of Nova-SN, but slightly faster than that of Nova-CN. From the left plot of Fig. 2, when there are 100 tenants in the Koala-based cloud, the load ratio of compute nodes is 0.54. For the Nova-SN, WRR and Nova-CN methods, the load ratio of compute nodes equals to 0.65, 0.62 and 0.53, respectively. Fig. 3 shows the CDF of the load ratio of compute nodes. We observe that when using Nova-CN and R-VMP-AI methods, there are



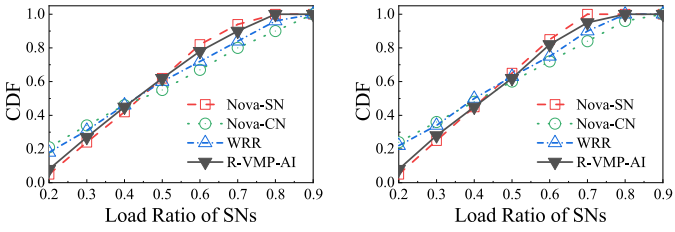
(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 3. CDF versus load ratio of compute nodes.



(a) Koala-Based Cloud (b) Google-Based Cloud

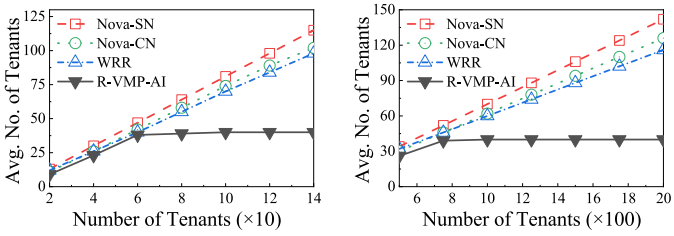
Fig. 4. Load ratio of service nodes versus number of tenants.



(a) Koala-Based Cloud (b) Google-Based Cloud

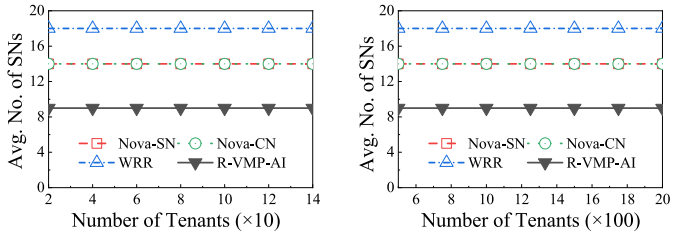
Fig. 5. CDF versus load ratio of service nodes.

no compute nodes in the Google-Based cloud with a load rate greater than 0.7. In contrast, when using Nova-SN and WRR methods, more than 15% of compute nodes have a load factor greater than 0.7. Figs. 2 and 3 show that R-VMP-AI can achieve better load balancing performance of compute nodes compared with Nova-SN and WRR, but slightly worse performance than Nova-CN. Since the goal of Nova-CN is only to achieve load balancing of compute nodes, this shows that our algorithm has a good load balancing performance of compute nodes. Then we observe the load ratio of service nodes. In Fig. 4, as the number of tenants increases, we find the increasing rate of R-VMP-AI is much slower than that of Nova-CN, but slightly faster than that of Nova-SN. When there are 1000 tenants in the Google-based cloud, the load ratio of service nodes is 0.38. For the Nova-SN, WRR and Nova-CN methods, the load ratio of compute nodes equals to 0.365, 0.4 and 0.43, respectively. Fig. 5 shows the CDF of the load ratio of service nodes, which provides an overview of resource usage. It is easy to observe that the load rate of service nodes in the cloud is more balanced when using Nova-SN and R-VMP-AI methods than when using Nova-CN and WRR



(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 6. Average number of tenants in each service node versus number of tenants.

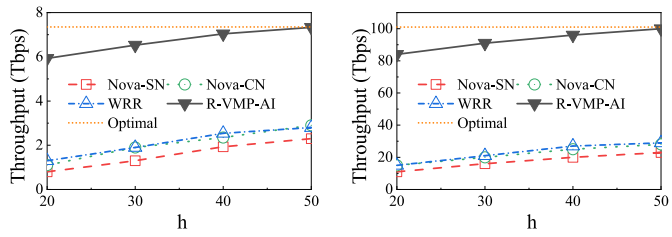


(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 7. Average Number of Service Nodes Served by Each Tenant versus Number of Tenants.

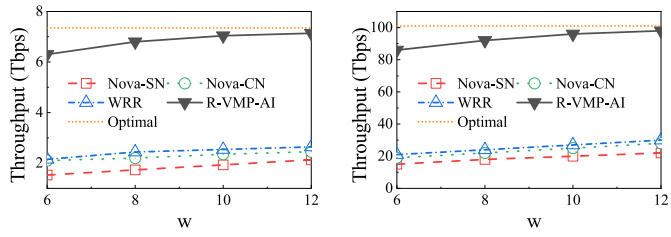
methods. Figs. 4 and 5 show that R-VMP-AI can achieve better load balancing performance of service nodes compared with WRR and LLF-CN, but slightly worse than LLF-SN. Since the goal of LLF-SN is only to achieve load balancing of service nodes, this shows that our algorithm has a good load balancing performance of service nodes. That is because the Nova-SN algorithm only balances the load among all service nodes. As shown in the right plot of Figs. 2 and 4, when there are 1500 tenants in the Google-based cloud, the load ratio of compute nodes and service nodes by Nova-CN are 0.53 and 0.63, respectively; the load ratio of compute nodes and service nodes by Nova-SN are 0.64 and 0.535, respectively. For the Nova-CN method, the load balancing factor is $\max\{0.53, 0.63\} = 0.63$. For Nova-SN, the load balancing factor is $\max\{0.64, 0.535\} = 0.64$. The load balancing factor λ of our algorithm is $\max\{0.545, 0.55\} = 0.55$. This shows our algorithm achieves a load balancing trade-off between service nodes and compute nodes, which combines the advantages of Nova-CN and Nova-SN. This is because we consider the load balancing of compute nodes and service nodes as a whole, which plays an important role in the load balancing of the cloud.

The second set of simulations exhibits the scope of negative impact on the cloud network when abnormal events occur. Specifically, we use the average number of tenants served by each service node and the average service node accessed as the impact scope of service node failures and malicious tenants, respectively. In Fig. 6, as the number of tenants increases, we find that the average number of served tenants by each service node accordingly increases by three benchmarks. However, the number of tenants served by one service node is not more than 40 by R-VMP-AI. This means once a service node fails, at most 40 tenants will be affected. Compared with the other three benchmarks, R-VMP-AI limits the scope of service node failures. For example, when there are 140 tenants in the Koala-based cloud, the number of tenants in one service node are 102, 98 and 115 by Nova-CN, WRR and Nova-SN, respectively. Based on the above analysis, our algorithm decreases the impact scope of service node failures by 60.7%, 59.1%, and 65.2%, respectively. Then we observe the impact of malicious tenants on clouds. In Fig. 7, since each tenant requests 1-50 VMs, the number of service nodes accessed by each tenant is relatively fixed. We find that a tenant only accesses ten service nodes by R-VMP-AI. This



(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 8. Throughput of All Service Nodes versus h.

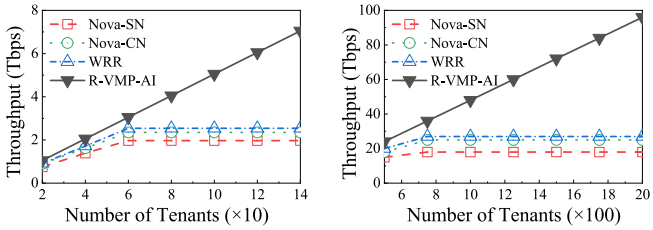


(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 9. Throughput of all service nodes versus w.

means when one malicious tenant attacks the cloud, at most ten service nodes will be attacked. In the Koala-based cloud, once a malicious tenant attacks the cloud, the number of average attacked service nodes are 9, 14, 14, 18 by R-VMP-AI, Nova-SN, Nova-CN, and WRR, respectively. Compared with Nova-SN, Nova-CN, and WRR, our algorithm decreases the impact scope of malicious tenants by 35.7%, 35.7% and 50%, respectively. Based on the above analysis, our algorithm not only achieves resource load balancing, but also limits the network impact scope of the two above abnormal situations.

The third set of simulations shows the impact of the setting of two system parameters (h and w) on the valid throughput of clouds. For convenience, we add the throughput of the cloud without considering the service node constraints and tenant constraints in Figs. 8 and 9 and name it as "Optimal" to show how our proposed constraints limit the system throughput. As shown in Figs. 8 and 9, the throughput achieved by our proposed algorithm is much higher than that of Nova-SN, Nova-CN, and WRR,



(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 10. Throughput of All Service Nodes versus Number of Tenants.

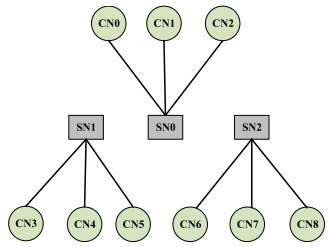


Fig. 11. The topology of a small-scale experiments. It contains 9 compute nodes (CNs) and 3 service nodes (SNs).

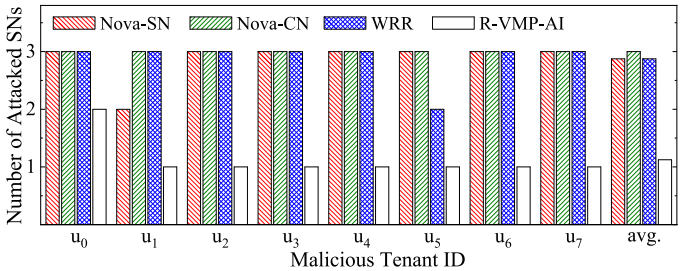


Fig. 12. The number of attacked service nodes when the cloud is attacked by a malicious tenant.

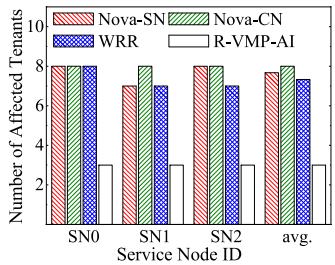


Fig. 13. The number of affected tenants when the cloud faces one SN failure.

regardless of the values of h and w. For example, in Fig. 8(a), when h = 40, the valid throughput of the cloud are 2.35, 2.54, 1.93 and 7.04 Tbps by Nova-CN, WRR, Nova-SN and R-VMP-AI, respectively. R-VMP-AI improves the throughput by 199%, 177% and 267% compared with Nova-CN, WRR and Nova-SN, respectively. In Fig. 9(a), when w = 12, the valid throughput of the cloud are 2.45, 2.64, 2.13 and 7.14 Tbps by Nova-CN, WRR, Nova-SN and R-VMP-AI, respectively. R-VMP-AI improves the valid throughput by 191%, 170% and 235% compared with Nova-CN, WRR and Nova-SN, respectively. The reason is that the other three benchmarks will abandon many VMs to meet the service node constraints and the tenant constraints, while R-VMP-AI already takes service node and tenant constraints into account when deploying VMs. In addition, the performance of our algorithm is close to the Optimal value, especially with the increase of h. For example, in Fig. 8, when h=40 (50) in the Google-based cloud, our algorithm only loses 4.5% (0.9%) throughput compared with Optimal.

Our last set of simulations compares the valid throughput of the cloud by changing the number of tenants in clouds and the results are shown in Fig. 10. After placing all VMs on the compute nodes, we consider whether the traffic served by service nodes will violate the service node constraints and the tenant constraints. If the number of tenants in one service node exceeds 40, the service node will refuse to serve the extra violated traffic. If one tenant can access more than 10 service nodes, only part of these service node can hanlde the tenant's traffic. In Fig. 10, when the number of tenants is small, the proportion of violating traffic is also small, the throughput is still increasing. However, When there are too many tenants, the traffic cannot be served. For example, when there are 140 tenants in the Koala-based cloud, the cloud throughput equals to 5.05, 1.97, 2.54, and 2.35 Tbps by R-VMP-AI, Nova-SN, WRR, and Nova-CN, respectively. Our algorithm improves the cloud throughput by 150% on average.

According to the simulation results, we conclude three conclusions. First, by Figs. 2, 3, 4, compared with the state-of-art algorithms, our algorithm can simultaneously achieve better load balancing among all compute nodes and among all service nodes. Second, by Figs. 6 and 7, our algorithm reduces the impact scope of service node failures and malicious tenants by about 60% and 40%, respectively. Third, by Figs. 8, 9, 10, under the service node constraints and the tenant constraints, our algorithm can improve the cloud throughput by about 150%.

C. System Implementation in a Small-Scale Testbed

In this section, we implement small-scale experiments with the popular OpenStack architecture [44]. The small-scale cloud consists of nine compute nodes and three service nodes, and each service node serves the traffic from three compute nodes, as depicted in Fig. 11. Each compute node or service node runs on a single server with a core i5-10400 processor with 12 vCPUs and 16 GB of RAM. The peak traffic processing capacity of each service node is set as 1000 Mbps. In our experiments, we deploy eight tenants named $u_0, u_1, \ldots, u_7$, and each tenant randomly requests a set of VMs from the cloud. We use iperf3 [55] to generate the normal TCP service traffic between VMs and the service nodes, and generate the malicious traffic by the hping tool [56]. In our experiments, when a service node fails, we use the classic backup method [40] to transfer the traffic served by the failed service node to other available backup nodes. When a tenant's traffic is served by a failed service node, the tenant's makespan will increase significantly.

1) The Scope of Abnormal Events: In the first set of experiments, we observe the impact scope of malicious tenants and service node failures, as shown in Figs. 12 and 13. In Fig. 12, we

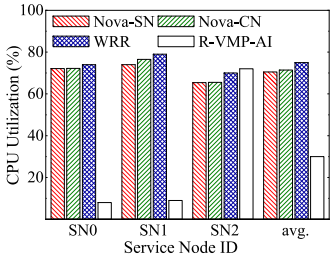


Fig. 14. The CPU utilization of SNs when the cloud is attacked by a malicious tenant.

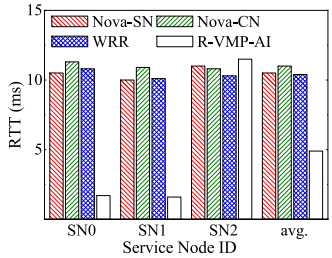


Fig. 16. The round-trip time (RTT) of SNs when the cloud is attacked by a malicious tenant.

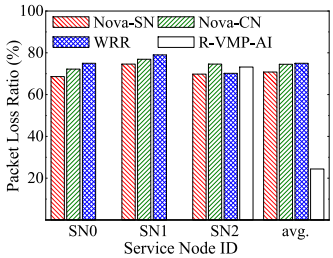


Fig. 15. The packet loss ratio of SNs when the cloud is attacked by a malicious tenant.

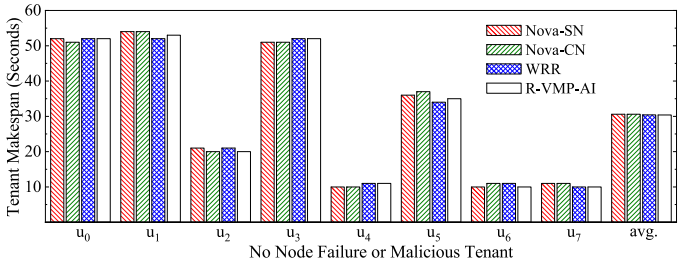


Fig. 17. Tenant task makespan versus Tenant ID without node failure and malicious tenant.

count the number of attacked service nodes when there exists one malicious tenant in the cloud. If there exists one malicious tenant $u \in \{u_0, u_2, u_3, u_4, u_6, u_7\}$, all three service nodes will be attacked for the three benchmarks. However, we find that R-VMP-AI can decrease the number of attacked service nodes from three to one in most cases. We observe that R-VMP-AI reduces the average impact scope of malicious tenants by 61%, 63% and 61% compared with Nova-CN, WRR and Nova-SN, respectively. In Fig. 13, we observe the number of tenants affected by service node failures. By Nova-CN, the failure of any service node will cause all eight tenants to suffer QoS degradation. By WRR and Nova-SN, the number of affected tenants by the failure of the three service nodes equals 8, 7, 7 and 8, 7, 8, respectively. However, we find that R-VMP-AI can decrease the number of affected tenants to 3. We can see that R-VMP-AI decreases the average impact scope of service node failures by 62.5%, 59.1% and 60.9% compared with Nova-CN, WRR and Nova-SN, respectively. From the above experimental results, we conclude that compared with three benchmarks, R-VMP-AI can simultaneously decrease the impact scope of malicious tenants and service node failures through two novel constraints.

2) Service Node Performance: The second set of experiments measures the service node performance when there exists one malicious tenant, as shown in Figs. 14, 15, and 16. Specifically, we randomly choose one tenant from the eight tenants as the malicious one, which sends a large amount of malicious traffic through hping3 [56] to simulate attacks, and measure the CPU utilization, the round-trip time (RTT), and the packet loss ratio of the three service nodes. In Fig. 14, we find that when the service node is not attacked, the CPU utilization is about 6%-9%. When the service node is under attack, its CPU utilization may reach 70% or even higher. R-VMP-AI reduces the average CPU utilization by 57.4%, 57.9% and 60.2% compared with Nova-CN, WRR, and Nova-SN, respectively. This is because R-VMP-AI limits the number of attacked service nodes, malicious tenants only attack a small part of the service nodes, and most of the service nodes still works normally. In Fig. 15, we observe that the service node will not lose packets when it is not attacked, but the packet loss rate reaches 75% when it is under attack. In Fig. 15, R-VMP-AI reduces the packet loss ratio by 67.2%, 67.5% and 65.8% compared with Nova-CN, WRR, and Nova-SN, respectively. In Fig. 16, we demonstrate the RTT of all the service nodes. When the service node runs normally, the RTT is about 1.4ms-1.8 ms, and increases to 10ms-11 ms when it is attacked. Moreover, R-VMP-AI achieves the less average RTT compared with other algorithms. For example, the average RTT results are 11 ms, 10.4 ms, 10.5 ms and 4.9 ms corresponding to Nova-CN, WRR, Nova-SN and R-VMP-AI, respectively. This means that R-VMP-AI decreases the RTT by 55.4%, 52.8% and 53.3% compared with Nova-CN, WRR, and Nova-SN, respectively. From the above experimental results, R-VMP-AI limits the number of service nodes that a malicious tenant can attack, and achieve much better network performance (eg., CPU utilization, packet loss ratio and RTT).

3) Tenant Task Makespan Performance: Finally, we test the task makespan performance of each tenant, as shown in Figs. 17, 18, and 19. To measure the task makespan, each tenant generates 50 subtasks and sends them to the fixed VM instances one by one. The data size of these 50 subtasks satisfy the 2/8 distribution, that is, 10 subtasks are accounting for 80% data. The remaining 40 mice subtasks account for the remaining 20% data [61]. We mainly test the tenant makespan in three scenarios. The first scenario is that there is no node failure nor malicious tenants

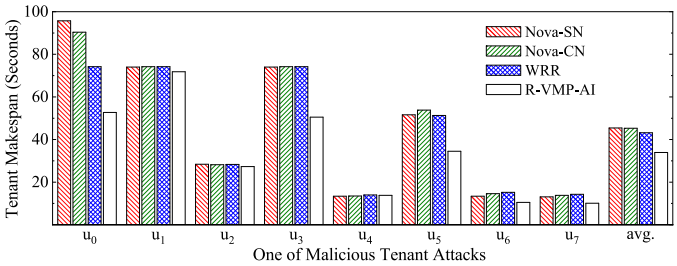


Fig. 18. Tenant task makespan versus Tenant ID when the cloud is attacked by one malicious tenant.

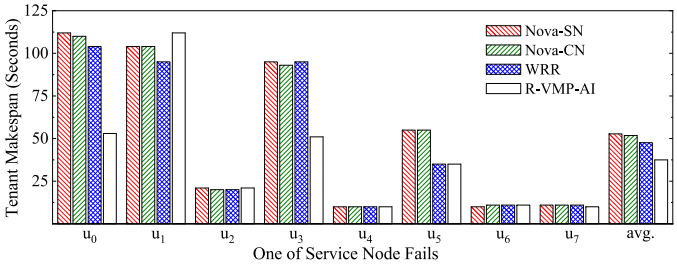


Fig. 19. Tenant task makespan versus Tenant ID when the cloud is facing one of service node failure.

in the cloud. In the second scenario, there is a malicious tenant attack in the cloud. The last scenario is that one service node fails. In Fig. 17, we first measure the task makespan of each tenant when the cloud runs normally. We find that the task makespan of each tenant is approximately the same. For example, the makespan of tenant u_0 is 51 s, 52 s, 52 s and 52 s by Nova-CN, WRR, Nova-SN and R-VMP-AI, respectively. In Fig. 18, we observe the makespan increases when the cloud is under the attack of one malicious tenant. Specifically, the tenant makespan has increased more drastically for Nova-CN and Nova-SN. For example, the task makespan of tenant u_0 increases to 90.4 s and 95.7 s using Nova-CN and Nova-SN, respectively. However, the task makespan of tenant u_0 keeps 52 s using our algorithm. This is because the malicious tenant's attack cannot reach the service node accessed by tenant u_0. Last, we collect the task makespan when the cloud occurs service node failure, as shown in Fig. 19. We observe that our algorithm is more resistant to service node failure. For example, only tenant u_1 experiences a significant increase in the makespan metric. When a service node fails, most tenants still perform their tasks normally. Compared with the state-of-art solutions, our algorithm decreases 25% tenant task makespan on average.

According to the above experimental results, we conclude that R-VMP-AI can limit the impact scope of malicious tenants and service node failures.

VI. Conclusion

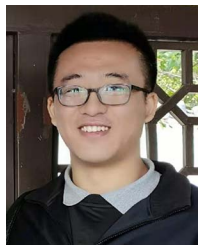
How to alleviate the negative impact scope when an abnormal event (eg., malicious tenants and node failures) occurs is a critical challenge in clouds. In this paper, we propose two novel constraints (ie., the service node constraints and the tenant constraints), and a scheme to alleviate the negative impact range of abnormal events through efficient multi-constraint VM placement without consuming additional resources. We formulate the multi-constraint VM placement as an NP-hard problem (VMP-AI) and design a rounding-based algorithm (R-VMP-AI) with bounded approximation factors to solve it. We implement our proposed algorithm on a physical testbed. Both the experimental results and simulation results show that our proposed algorithm can significantly reduce the negative impact of abnormal events without affecting resource balance compared with existing solutions. For example, our algorithm reduces the impact scope of service node failure by 60%, the impact scope of malicious tenants by 40%, and the tenant task makespan by 25% compared with other alternatives.

References

[1] Y. Zhai, G. Zhao, H. Xu, Y. Zhao, J. Liu, and X. Fan, "Towards robust multi-tenant clouds through multi-constrained VM placement," in *Proc. IEEE/ACM 29th Int. Symp. Qual. Service*, 2021, pp. 1–6.
[2] Amazon EC2, Accessed: Aug. 10, 2021. [Online]. Available: https://docs.aws.amazon.com/ec2/index.html
[3] "Alibaba cloud," Accessed: Aug. 10, 2021. [Online]. Available: https://us.alibabacloud.com
[4] M. T. Arashloo, P. Shirshov, R. Gandhi, G. Lu, L. Yuan, and J. Rexford, "A scalable VPN gateway for multi-tenant cloud services," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 1, pp. 49–55, 2018.
[5] M. Rahman, S. Iqbal, and J. Gao, "Load balancer as a service in cloud computing," in *Proc. IEEE 8th Int. Symp. Service Oriented System Eng.*, 2014, pp. 204–211.
[6] H. Jia et al., "Security strategy for virtual machine allocation in cloud computing," *Procedia Comput. Sci.*, vol. 147, pp. 140–33, 2019.
[7] S. Zhang et al., "Efficient and robust syslog parsing for network devices in datacenter networks," *IEEE Access*, vol. 8, pp. 30245–33, 2020.
[8] N.-N. Dao et al., "Securing heterogeneous IoT with intelligent DDoS attack behavior learning," *IEEE Syst. J.*, vol. 16, no. 2, pp. 1974–1983, Jun. 2022.
[9] J. Hou, P. Fu, Z. Cao, and A. Xu, "Machine learning based DDoS detection through netflow analysis," in *Proc. IEEE Mil. Commun. Conf.*, 2018, pp. 1–6.
[10] A. O. F. Atya, Z. Qian, S. V. Krishnamurthy, T. L. Porta, P. McDaniel, and L. Marvel, "Malicious co-residency on the cloud: Attacks and defense," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[11] B. Arzani et al., "PrivateEye: Scalable and privacy-preserving compromise detection in the cloud," in *Proc. 17th USENIX Symp. Networked Syst. Des. Implementation*, 2020, pp. 797–815.
[12] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 337–350, Fourth Quarter 2010.
[13] I. P. Egwutuoha, S. Chen, D. Levy, and B. Selic, "A fault tolerance framework for high performance computing in cloud," in *Proc. IEEE/ACM 12th Int. Symp. Cluster Cloud Grid Comput.*, 2012, pp. 709–710.
[14] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: A field study of middlebox failures in datacenters," in *Proc. Conf. Internet Meas. Conf.*, 2013, pp. 9–22.
[15] C. Tan et al., "NetBouncer: Active device and link failure localization in data center networks," in *Proc. 16th USENIX Symp. Networked Syst. Des. Implementation*, 2019, pp. 599–614.
[16] P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "Intrusion detection techniques in cloud environment: A survey," *J. Netw. Comput. Appl.*, vol. 77, pp. 18–33, 2017.
[17] B. Yong, G. Zhang, H. Chen, and Q. Zhou, "Intelligent monitor system based on cloud and convolutional neural networks," *J. Supercomputing*, vol. 73, no. 7, pp. 3260–3276, 2017.
[18] F. L. Pires and B. Barán, "A virtual machine placement taxonomy," in *Proc. IEEE/ACM 15th Int. Symp. Cluster Cloud Grid Comput.*, 2015, pp. 159–168.
[19] S. Farzai, M. H. Shirvani, and M. Rabbani, "Multi-objective communication-aware optimization for virtual machine placement in cloud datacenters," *Sustain. Comput.: Inform. Syst.*, vol. 28, 2020, Art. no. 100374.

[20] A. Shieh, S. Kandula, A. G. Greenberg, and C. Kim, "Seawall: Performance isolation for cloud datacenter networks," in *Proc. 2nd USENIX Conf. Hot Top. Cloud Comput.*, 2010, p. 1.
[21] R. Birke, A. Podzimek, L. Y. Chen, and E. Smirni, "State-of-the-practice in data center virtualization: Toward a better understanding of vm usage," in *Proc. IEEE/IFIP 43rd Annu. Int. Conf. Dependable Syst. Netw.*, 2013, pp. 1–12.
[22] S. Yang, P. Wieder, R. Yahyapour, S. Trajanovski, and X. Fu, "Reliable virtual machine placement and routing in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2965–2978, Oct. 2017.
[23] OpenStack Docs: Scheduling, Accessed: Nov. 20, 2022. [Online]. Available: https://docs.openstack.org/mitaka/config-reference/compute/scheduler.html
[24] M. Mishra and A. Sahoo, "On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach," in *Proc. IEEE 4th Int. Conf. Cloud Comput.*, 2011, pp. 275–282.
[25] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. IEEE INFOCOM*, 2012, pp. 2876–2880.
[26] A. Lebre, J. Pastor, A. Simonet, and M. Südholt, "Putting the next 500 VM placement algorithms to the acid test: The infrastructure provider viewpoint," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 204–217, Jan. 2019.
[27] H. Shen and L. Chen, "CompVM: A complementary VM allocation mechanism for cloud systems," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1348–1361, Jun. 2018.
[28] C. Delimitrou and C. Kozyrakis, "Bolt: I know what you did last summer... in the cloud," *ACM SIGARCH Comput. Architecture News*, vol. 45, no. 1, pp. 599–613, 2017.
[29] B. Grobauer, T. Walloschek, and E. Stocker, "Understanding cloud computing vulnerabilities," *IEEE Secur. Privacy*, vol. 9, no. 2, pp. 50–57, Mar./Apr. 2011.
[30] R. Hofstede, L. Hendriks, A. Sperotto, and A. Pras, "SSH compromise detection using NetFlow/IPFIX," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 20–26, 2014.
[31] J. Cao, B. Yu, F. Dong, X. Zhu, and S. Xu, "Entropy-based denial-of-service attack detection in cloud data center," *Concurrency Computation: Pract. Experience*, vol. 27, no. 18, pp. 5623–5639, 2015.
[32] H. P. K. Tiwari and A. Chaudhary, "Secure VM placement analysis against co-location based attack in cloud," *J. Discrete Math. Sci. Cryptogr.*, vol. 24, no. 5, pp. 1457–1465, 2021.
[33] W. Ding et al., "DFA-VMP: An efficient and secure virtual machine placement strategy under cloud environment," *Peer-to-Peer Netw. Appl.*, vol. 11, no. 2, pp. 318–333, 2018.
[34] Y. Liu, X. Ruan, S. Cai, R. Li, and H. He, "An optimized VM allocation strategy to make a secure and energy-efficient cloud against co-residence attack," in *Proc. Int. Conf. Comput. Netw. Commun.*, 2018, pp. 349–353.
[35] Y. Azar, S. Kamara, I. Menache, M. Raykova, and B. Shepard, "Co-location-resistant clouds," in *Proc. 6th Ed. ACM Workshop Cloud Comput. Secur.*, 2014, pp. 9–20.
[36] M. Li, Y. Zhang, K. Bai, W. Zang, M. Yu, and X. He, "Improving cloud survivability through dependency based virtual machine placement," in *Proc. Int. Conf. Secur. Cryptograph*, 2012, pp. 321–326.
[37] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 350–361.
[38] A. Engelmann and A. Jukan, "A reliability study of parallelized VNF chaining," in *Proc. IEEE Int. Conf. Commun.*, 2018, pp. 1–6.
[39] J. Li, W. Liang, M. Huang, and X. Jia, "Reliability-aware network service provisioning in mobile edge-cloud networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1545–1558, Jul. 2020.
[40] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme," *IEEE Trans. Mobile Comput.*, vol. 21, no. 8, pp. 2994–3008, Aug. 2022.
[41] J. Zhang, Z. Wang, C. Peng, L. Zhang, T. Huang, and Y. Liu, "RABA: Resource-aware backup allocation for a chain of virtual network functions," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1918–1926.
[42] Y. Li et al., "Predicting node failures in an ultra-large-scale cloud computing platform: An AIOps solution," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 2, pp. 1–24, 2020.
[43] L. Caviglione, M. Gaggero, M. Paolucci, and R. Ronco, "Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters," *Soft Comput.*, vol. 25, no. 19, pp. 12569–12588, 2021.
[44] "Build the future of open infrastructure," Accessed: Aug. 10, 2021. [Online]. Available: https://openstack.org
[45] S. Sotiriadis, N. Bessis, and R. Buyya, "Self managed virtual machine scheduling in cloud systems," *Inf. Sci.*, vol. 433, pp. 381–33, 2018.
[46] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.
[47] S. Sotiriadis, N. Bessis, C. Amza, and R. Buyya, "Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling," *IEEE Trans. Serv. Comput.*, vol. 12, no. 2, pp. 319–334, Mar./Apr. 2016.
[48] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proc. 11th Int. Conf. Netw. Service Manage.*, 2015, pp. 50–56.
[49] G. C. Fox, J. Qiu, S. Kamburugamuve, S. Jha, and A. Luckow, "HPC-ABDS high performance computing enhanced apache big data stack," in *Proc. 15th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2015, pp. 1057–1066.
[50] W. JingZhou, Z. Gongming, X. Hongli, H. He, L. Luyao, and Y. Yongqiang, "Robust service mapping in multi-tenant clouds," in *Proc. 40th Annu. IEEE Int. Conf. Comput. Commun.*, 2021, pp. 1–11.
[51] J. K. Lenstra, D. B. Shmoys, and É. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Math. Program.*, vol. 46, no. 1, pp. 259–271, 1990.
[52] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Proc. 16th Annu. Symp. Found.s Comput. Sci.*, 1975, pp. 184–193.
[53] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas, "Fast approximation algorithms for multicommodity flow problems," *J. Comput. System Sci.*, vol. 50, no. 2, pp. 228–243, 1995.
[54] P. Raghavan and C. D. Tompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.
[55] "iPerf - the ultimate speed test tool for TCP, UDP and SCTP," Accessed: Aug. 10, 2021. [Online]. Available: https://iperf.fr
[56] "hping - the TCP/IP packet assembler/analyzer," Accessed: Aug. 10, 2021. [Online]. Available: https://github.com/antirez/hping
[57] N. K. C. Das, M. S. George, and P. Jaya, "Incorporating weighted round robin in honeybee algorithm for enhanced load balancing in cloud environment," in *Proc. Int. Conf. Commun. Signal Process.*, 2017, pp. 0384–0389.
[58] "Tencent cloud instance types," Accessed: Aug. 10, 2021. [Online]. Available: intl.cloud.tencent.com/document/product/213/11518
[59] C. Baun and M. Kunze, "The koala cloud management service: A modern approach for cloud infrastructure management," in *Proc. 1st Int. Workshop Cloud Comput. Platforms*, New York, NY, USA, 2011.
[60] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format+ schema," *Google Inc.*, White Paper, pp. 1–14, 2011.
[61] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas.*, 2009, pp. 202–208.

Gongming Zhao (Member, IEEE) received the PhD degree in computer software and theory from the University of Science and Technology of China, in 2020. He is currently an associate professor with the University of Science and Technology of China. His current research interests include software-defined networks and cloud computing.



Jiawei Liu received the BS degree from the College of Computer Science and Technology, Jilin University, in 2014. He is currently working toward the EngD degree in computer technology with the University of Science and Technology of China. His current research interests include software-defined networks, cloud computing, and programmable networks.

Yutong Zhai received the BS degree in computer science and the PhD degree in computer software and theory from the University of Science and Technology of China, China, in 2017 and 2022, respectively. His research interests include the areas of software-defined networking, internet traffic measurement, and edge computing.

Hongli Xu (Member, IEEE) received the BS degree in computer science and the PhD degree in computer software and theory from the University of Science and Technology of China, China, in 2002 and 2007, respectively. He is currently a professor with the School of Computer Science and Technology, University of Science and Technology of China (USTC). He has published more than 100 articles in famous journals and conferences, including *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Parallel and Distributed Systems*, International Conference on Computer Communications (INFOCOM), and International Conference on Network Protocols (ICNP). He has held more than 30 patents. His research interests include software defined networks, edge computing, and the Internet of Thing. He was awarded the Outstanding Youth Science Foundation of NSFC in 2018. He has won the best paper award or the best paper candidate in several famous conferences.

Huang He (Member, IEEE) received the PhD degree from the School of Computer Science and Technology, University of Science and Technology of China (USTC), in 2011. He is currently a professor with the School of Computer Science and Technology, Soochow University, China. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a member of the ACM.