Towards Robust Multi-Tenant Clouds Through Multi-Constrained VM Placement

Yutong Zhai[1,2] Gongming Zhao[1,2] Hongli Xu[1,2] Yangming Zhao[1,2] Jiawei Liu[2,3] Xingpeng Fan[1,2]
[1]School of Computer Science and Technology, University of Science and Technology of China, China
[2]Suzhou Institute for Advanced Study, University of Science and Technology of China, China
[3]School of Software Engineering, University of Science and Technology of China, China

***Abstract*—More and more tenants (enterprises and personal users) migrate their tasks to clouds since it is a simple and low-cost way to obtain enough computing resources. However, due to potential node failures and malicious tenants, the modern cloud encounters one critical challenge, *i.e.*, robustness. Conventionally, the cloud vendors deploy auxiliary systems to protect the cloud, which requires additional resource cost and increases the network complexity. To enhance the system robustness, this paper proposes a complementary scheme to improve the cloud robustness through efficient VM placement. Specifically, to alleviate the impact of malicious tenants and node failures on the cloud, when deploying VMs, we limit the number of pods (or service nodes) that each tenant can access, and the number of tenants hosted by each pod (or service node). Though there are a lot of works on VM placement, it is very challenging when the robustness issue is taken into consideration. To solve this problem, we formulate an integer linear programming and propose a rounding-based algorithm with a logarithmic approximation ratio. The simulation results show the high efficiency of the proposed algorithm. For example, our algorithm can improve the network throughput by 150% with other alternatives.**

***Index Terms*—Multi-Tenant Clouds, Robustness, Service Node, Computing Node, VM Placement.**

I. Introduction

With the development of cloud computing, it has become common for enterprises to migrate their computation tasks to the cloud since such a migration significantly reduces the complexity and costs of managing private data centers. As shown in Fig. 1, a typical cloud (*e.g.*, Amazon EC2 [1] and Alibaba cloud [2]) is composed with many *pods*, each of which consists of a set of *computing nodes* and one *service node* [3]. Cloud vendors deploy a huge number of computing nodes in their clouds and provide computing resources (*e.g.*, CPU and RAM) to tenants in the form of VMs. On the other hand, a service node in each pod can provide various network services (*e.g.*, VPN and ELB) to tenants.

In multi-tenant clouds, malicious tenants and service node failures are very common [4], which brings the robustness issue. Specifically, malicious tenants will launch wide spectrum network attacks, including denial of service

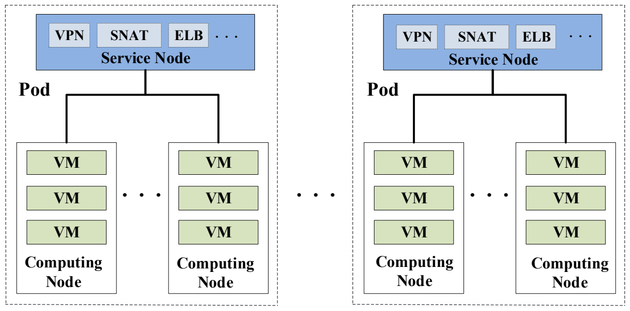


Fig. 1: A typical cloud architecture consists of several pods, and each pod includes a set of computing nodes and a service node [3].

attacks (DoS) and co-residency attacks [5] against the service node. For example, Amazon EC2 clouds are frequently attacked by spammers and DoS [6]. Malicious tenants send a large amount of traffic, which paralyzes service nodes and decreases network performance. Once a service node is defeated by a malicious tenant, all the VMs in the same pod cannot get any be served network services provided by this service node. Meanwhile, the service nodes (all the network functions residing with these service nodes) themselves may frequently fail in commodity clouds [4]. According to a recent report [7], the median time between two consecutive failures of firewall and load balancer is 7.5 hours and 5.2 hours, respectively. These service node failures degrade the QoS provided to tenants [8].

Conventionally, the cloud vendors address the robustness issue by deploying accessorial systems, such as an intrusion detection system to protect service nodes against malicious tenants [9] and a monitoring system to monitor the service node status [10]. Though this method is feasible to promote system robustness, additional resources are required, and the system management will be more complexity. In this work, we argue that cloud robustness can be promoted through multi-constraint VM placement without consuming additional resources. It is worth noting that our work plays a parallel story to further promote cloud robustness, rather than trying to improve previous methods or even substitute them.

To this end, we will introduce two constraints when we deploy VMs in the cloud, besides purely pursuing load balancing on computing nodes and service nodes. First, the VMs belonging to the same tenants can be allocated in most h pods, and second, each pod can host VMs renting to at most w tenants, where h and w are parameters determined by the system. The rationale behind the first constraint is that with proper isolation techniques [11], a malicious tenant can only attack the service nodes that provide services for the allocated VMs. Hence the first constraint can limit the damage caused by a single malicious tenant. On the other hand, the failure of a service node will result in performance degradation to all the tenants in the corresponding pod. Therefore, we introduce the second constraint.

Note that, in this work, we do not consider the failure of computing nodes, since a computing node usually only supports a limited number of VMs. For example, the work [12] shows that each computing node supports 10.8 VM instances on average. Thus, the failure of a computing node cannot significantly degrade the performance from the cloud's perspective. Actually, the algorithm proposed in our work can also be easily extended to the case that the failure of computing nodes should be taken into considered. The main contributions of this paper are as follows:

1) We formally formulate the robust VM placement problem, named VMPR, and analyze the problem complexity.

2) To solve VMPR, we propose an efficient algorithm called R-VMPR, and analyze the approximate performance.

3) We evaluate the proposed method through large-scale simulations. The simulation results show the high efficiency of our proposed algorithm.

The rest of this paper is organized as follows. Section II describes the system model. Section III formally defines the VMPR problem. In Section IV, we present an efficient algorithm to solve VMPR, and give the approximate performance. Section V shows the simulation results of our proposed algorithm, compared to some state-of-art solutions. In section VI, we conclude this paper.

II. System Model

A. System Model

1) Infrastructure Model: A typical cloud consists of many pods, each of which consists of a set of computing nodes and one service node [13]. Computing nodes provide computing resources to tenants in the form of VMs. We use $\mathcal{V} = \{v_1, v_2, ...v_n\}$ to denote the set of computing nodes, where n is the number of computing nodes. Service nodes provide various network services (*e.g.*, elastic load-balancing (ELB), firewall [14]) to tenants. In many common cloud architectures [3], each computing node is served by one service node, and each service node is responsible for providing services for VMs from all connected computing nodes. Since each service node provides several specific services [15], different types of service nodes are independent of each other. For ease of description, we only consider one type of service node in this paper. We denote the service node set as $\mathcal{S} = \{s_1, s_2, ...s_q\}$, where q is the number of pods in the cloud.

We use $s(v)$ to represent the service node that provides service for computing node $v \in \mathcal{V}$. In other words, service node $s(v)$ is responsible for providing services for VMs in computing 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 $z(u, s)$ be a binary constant which is 1 if service node s has background traffic from tenant u; otherwise $z(u, s) = 0$. For each computing node v, we use $R(v)$ to denote the total amount of resources (*e.g.*, RAM and CPU), and $b(v)$ to denote the amount of used resources. Note that, $R(v)$ and $b(v)$ can be expanded into resource vectors that represents different types of resources in computing 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 $\mathcal{U} = \{u_1, u_2, ...u_m\}$ denote the set of tenants, where $m = |\mathcal{U}|$ 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. Therefore, the required resources of each VM is different. For each tenant $u \in \mathcal{U}$, we represent the set of requested VM instances as $\mathcal{P}_u = \{p_{u,1}, p_{u,2}, ...p_{u,l_u}\}$, where l_u is the number of VM instances required by tenant u. Each VM instance $p_{u,d} \in \mathcal{P}_u$ ($1 \leq d \leq l_u$) will consume some computing resources, such as 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 $p_{u,d}$ [16].

III. Problem Definition and Complexity Analysis

A. Problem Definition

According to the illustration in section II-A, we formally define the VM Placement problem that can achieve cloud Robustness (VMPR) with the following three constraints: 1) To improve the robustness when encountering the service node failure event, we limit the number of tenants served by each service node, that is, each service node can serve h tenants at most; 2) To improve the robustness when encountering malicious tenants, we should ensure that the traffic of each tenant can be forwarded to w service nodes at most; 3) Moreover, we want to achieve the load balance among all service nodes and among all computing nodes. Accordingly, we formulate the VMPR problem as Eq. (1).

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

In Eq. (1), binary variable $x_v^{p_{u,d}}$ denotes whether VM instance $p_{u,d}$ of tenant u is placed in computing node v or not. Variable $y_s^u \in \{0,1\}$ 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 computing node. The second and third sets of inequalities express that once a VM of tenant u is placed on computing node v, the connected service node $s(v)$ will process the traffic from tenant u (*i.e.*, $y_{s(v)}^u = 1$). Similarly, if service node s has background traffic from tenant u (*i.e.*, $z(u,s) = 1$), we have $y_s^u = 1$. The fourth and fifth sets of inequalities represent the robustness constraints when encountering service node failures or malicious tenants. In other words, the number of service nodes accessed by each tenant cannot exceed h and the number of tenants that a service node can serve cannot exceed w. The sixth and seventh of inequalities indicates the resource load on each computing 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 computing nodes, that is, min λ.

IV. Algorithm Design

A. Algorithm Description

The R-VMPR algorithm starts by constructing the linear programming as relaxation of VMPR (LP-VMPR). More specifically, LP-VMPR assumes that both the VM placement and the VM's traffic demands are splittable. That is, LP-VMPR relaxes the variables $\{x_v^{p_{u,d}}\}$ and $\{y_s^u\}$ from integral to fractional. Since LP-VMPR 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 1-3), and the optimal result is denote as $\widetilde{\lambda}$. As LP-VMPR is relaxation of VMPR, $\widetilde{\lambda}$ is the lower-bound result for VMPR. The next step is to derive an integer solution by randomized rounding [17]. We derive the integer solution, denoted by $\{\widehat{x}_v^{p_{u,d}}\}$ and $\{\widehat{y}_s^u\}$. For each individual tenant u and service node s, R-VMPR rounds variable $\widehat{y}_s^u$ to 1 with probability $\widetilde{y}_s^u$ (lines 4-9) to keep the robustness constraints of service nodes. Each rounding decision is independent with each other. Then R-VMPR decides 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 computing node v (lines 10-17). Based on the above process, we get an integer solution $\{\widehat{x}_v^{p_{u,d}}, \widehat{y}_s^u\}$. That is, we obtain a VM placement solution that can achieve cloud robustness.

Algorithm 1 R-VMPR: Rounding-based Algorithm for VM-PR

```
1: Step 1: Sloving the relaxation of the VMPR Problem
2: Construct a linear programming named LP-VMPR formalized in Eq. (1)
3: Obtain the optimal fractional solutions $\{\widetilde{x}_v^{p_{u,d}}, \widetilde{y}_s^u\}$
4: Step 2: Selecting service nodes for tenants
5: for each tenant $u$ in $\mathcal{U}$ do
6:     for each service node $s$ in $\mathcal{S}$ do
7:         Set $\widehat{y}_s^u = 1$ with probability $\widetilde{y}_s^u$
8:         if $\widehat{y}_s^u == 1$ then
9:             Select service node $s$ for tenant $u$
10: Step 3: Placing VM instances on computing nodes
11: for each tenant $u$ in $\mathcal{U}$ do
12:     for each requested VM $p_{u,d}$ in $\mathcal{P}_u$ do
13:         for each computing node $v$ in $\mathcal{V}$ do
14:             if $\widehat{y}_{s(v)}^u == 1$ then
15:                 Set $\widehat{x}_v^{p_{u,d}} = 1$ with probability $\frac{\widetilde{x}_v^{p_{u,d}}}{\widetilde{y}_{s(v)}^u}$
16:                 if $\widehat{x}_v^{p_{u,d}} == 1$ then
17:                     Place VM $p_{u,d}$ on computing node $v$
```

Approximation Factor: The approximate factors of our algorithm are *bi-criteria approximations* with respect to both the objective value and robustness constraints. In many practical scenarios, these factors are constant. For example, consider a cloud with thousands of computing nodes and hundreds of service nodes. Then we have $n = 1000$ and $q = 100$. In general, one computing node can accommodate more than ten VMs, and there are more than ten computing nodes in one pod. Thus, we estimate the value $\alpha = 10$.

Then the bi-criteria approximation factor of computing n-odes and service nodes becomes $\frac{2\log n}{\alpha}+3=3.99$ and $\frac{2\log q}{\alpha}+3=3.66$, respectively.

V. Performance Evaluation

A. Performance Metrics and Benchmarks

1) Performance Metrics: We use the following five performance metrics to evaluate the robustness and load balancing performance of our proposed algorithm. (1) The load ratio of computing nodes (CNs); (2) The load ratio of service nodes (SNs); (3) The average number of tenants served by each service node; (4) The average number of service nodes that each tenant accesses; (5) The valid throughput of all service nodes. In the simulations, we use the first two metrics to evaluate the load balancing among all computing nodes and among all service nodes, respectively. The third to fifth metrics are used to evaluate robustness, that is, minimize the impact of malicious tenants and the service node failures. For each computing node, its resource load ratio is the maximum utilization of its CPU load ratio and RAM load ratio. We use the largest resource load ratio of all computing 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 their largest value as the second metric. Moreover, we also 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 throughput of all service nodes with the robustness constraints as the fifth metric.

2) Benchmarks: We choose three benchmarks for performance comparison. The first benchmark, called least-load-first with service nodes (LLF-SN) [18], represents a category solution that separately considers the remaining traffic processing capacity of service nodes and the remaining computing resource of computing nodes. LLF-SN first chooses one service node with the least workloads, and then selects the most idle computing node from the computing nodes served by this service node to place VM instances. The second benchmark is least-load-first with computing nodes (LLF-CN) [19], which places VM instances on the computing node with the least workloads. The last benchmark is the Weight-Round-Robin (WRR) [20] method. WRR first allocates weights to each service node based on its remaining resources and the remaining resources of the computing nodes served by it. Second, WRR generates a random value and selects the service node according to the value. Finally, WRR selects a computing node from the computing nodes served by the service node to place VM instances.

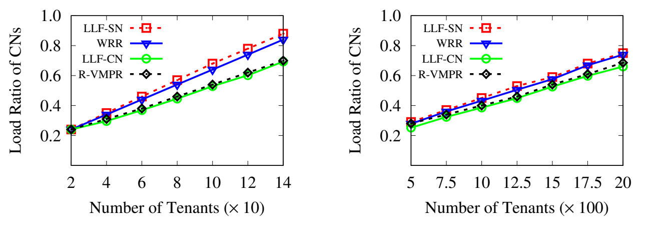
Name	vCPU	RAM(GB)	Bandwidth(Gbps)
s5.small2	1	2	1.5
s5.medium4	2	4	1.5
s5.large8	4	8	1.5
m5.small8	1	8	1.5
m5.medium16	2	16	1.5
m5.large32	4	32	1.5
c4.small2	1	2	3.0
c4.medium4	2	4	3.0
c4.large8	4	8	3.0

TABLE I: Multiple VM instances with detailed resources demand from Tencent Cloud [23].

B. Large-Scale Simulations

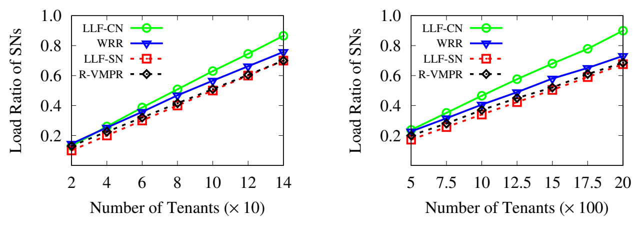
1) Simulation Settings: We perform our simulations over two infrastructure clouds. The first cloud is similar with Koala [21], which consists of 20 service nodes and 600 computing nodes, and each service node serves the traffic of 30 computing nodes. The second cloud is generated based on the Google cluster-data [22], which consists of 10047 computing nodes and 324 service nodes. Each service node serves the traffic of 25-35 computing 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 VM instances: standard, memory-optimized, and computing. The above three types of instances come from Tencent Clouds [23], even of which carries different amount of required resources (vCPU, RAM, and bandwidth), as shown in Table I. In Table I, the first three instances represent the standard instances, accommodating most applications, such as streaming media business and online-game [23]. The next three instances represent the memory-optimized instances, which are suitable for applications that require extensive memory operations, searches, and computations, such as high-performance databases and distributed memory cachings [23]. The last three instances are computing instances, which are suitable for compute-intensive workloads such as batch processing, high performance computing, and dedicated game servers [23]. The number of tenants in the above two clouds is set as 140 and 2000, respectively. Each tenant randomly creates 10-40 VM instances from Table I with different types. For each computing node, we set its vCPU cores and RAM capacity as 25 and 150GB, respectively. The traffic processing capacity of each service node is set as 500Gbps, and the robustness parameters h and w are set as 10 and 40 by default, respectively. We run each simulation 30 times and calculate the average value as the simulation results.

2) Simulation Results: In large-scale simulations, we first test the load balancing performance, then we evaluate



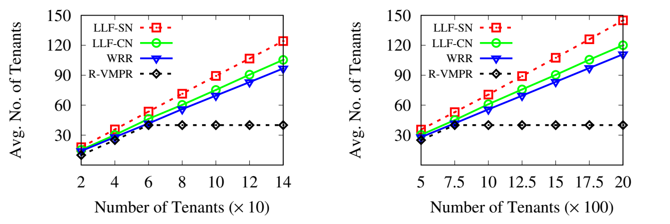
(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 2: Load Ratio of Computing Nodes vs. Number of Tenants



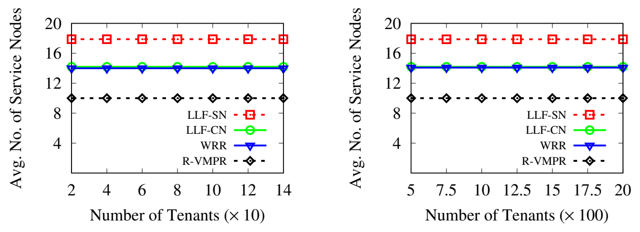
(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 3: Load Ratio of Service Nodes vs. Number of Tenants



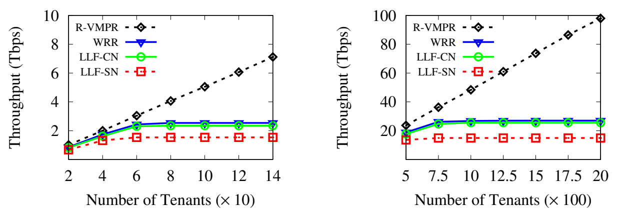
(a) Koala-Based Cloud (b) Google-Based Cloud

Fig. 4: Average Number of Tenants in Each Service Node vs. Number of Tenants



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

Fig. 5: Average Number of Service Nodes Served by Each Tenant vs. Number of Tenants



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

Fig. 6: Throughput of All Service Nodes vs. Number of Tenants

the robustness performance. The first set of simulations compares the load balancing performance of R-VMPR with three benchmarks. As shown in Figs. 2-3, when the number of tenants increases, the load ratio of computing nodes and service nodes accordingly increases in both the Koala-based cloud and the Google-based cloud. For the load ratio of computing nodes, the increasing rate of R-VMPR is much slower than that of both LLF-SN and WRR. From the left plot of Fig. 2, when there are 100 tenants in the Koala-based cloud, the load ratio of computing nodes is 0.54. For the LLF-SN, WRR and LLF-CN solutions, the load ratio of computing nodes equals to 0.68, 0.64 and 0.53, respectively. This shows R-VMPR can achieve better load balancing performance of computing nodes compared with LLF-SN and WRR, but slightly worse performance than LLF-CN. Since the goal of the LLF-CN algorithm is only to achieve load balancing among all computing nodes, this shows that our algorithm has good load balancing performance among computing nodes. For the load ratio of service nodes, we find the increasing rate of R-VMPR is much slower than that of both LLF-CN and WRR. Similarly, we find that R-VMPR can achieve better load balancing performance of service nodes compared with WRR and LLF-CN, but slightly worse than LLF-SN, as shown in Fig. 3. That is because the LLF-SN algorithm only balances the load among all service nodes. As shown in the right plot of Figs. 2-3, when there are 1500 tenants in the Google-based cloud, the load ratio of computing nodes and service nodes by LLF-CN are 0.52 and 0.68, respectively; the load ratio of computing nodes and service nodes by LLF-SN are 0.59 and 0.50, respectively. But the load balancing factor λ of our algorithm is $\max\{0.53, 0.53\} = 0.53$. Based on the above analysis, our algorithm can simultaneously achieve a load balancing trade-off among all service nodes and among all computing nodes, which combines the advantages of LLF-CN and LLF-SN. This is because we consider the load of computing nodes and service nodes as a whole, which plays an important role in load balancing of the cloud.

The second set of simulations exhibits the robustness metrics. We use the average number of tenants served by each service node and the average service node accessed by each tenant to evaluate the robustness of clouds, as shown in Figs. 4-5. As the number of tenants increases, we find that the average number of served tenants by each service node is not more than 40 by R-VMPR, as shown in Fig. 4. This means when a service node fails, at most 40 tenants will be affected. However, as the number of tenants increases, the number of affected tenants also increases by using the other three benchmarks. Moreover, we find that the failure of service nodes will affect more tenants in the Google-based cloud, as shown in Fig. 4(b). Compared with LLF-CN, WRR, and LLF-SN, our algorithm reduces the average number of tenants served by a service node by 62.6%, 55.6%, and 51.8%, respectively. Next we observe the impact of malicious tenants on clouds. In the Koala-based cloud, once a malicious tenant attacks the cloud, the number of average attacked service nodes is 10, 14.2, 14.0, 18.7 by R-VMPR, LLF-SN, LLF-CN, and WRR, respectively. Compared with SJF, WRR, and LLF-LF, our algorithm improves the robustness by 29.5%, 46.5% and 28.5%, respectively.

Our last set of simulations compares R-VMPR with other benchmarks under robustness constraints. By default, we set h as 10 and w as 40 both in the Koala-based cloud and the Google-based cloud. After placing all VMs on the computing nodes, we consider whether the traffic served by service nodes will violate the robustness constraints. If the service node violates the robustness constraints, then the service node will refuse to serve the extra violated traffic. As shown in the left plot of Fig. 6, when there are 100 tenants in the Koala-based cloud, the cloud throughput equals to 5.05, 1.53, 2.54, and 2.35 Tbps by R-VMPR, LLF-SN, WRR, and LLF-CN, respectively. Our algorithm improves the cloud throughput by 150% on average.

VI. Conclusion

Robustness is a critical challenge in clouds. In this paper, we improve the cloud robustness through multi-constraint VM placement, and we design a rounding-based algorithm with bounded approximation factors. The simulation results show that our proposed algorithm can achieve superior robustness performance compared with existing solutions.

Acknowledgement

The corresponding authors of this paper are Gongming Zhao and Hongli Xu. This article was supported in part by the National Science Foundation of China (NSFC) under Grants 61822210, 61936015, and U1709217; and in part by Anhui Initiative in Quantum Information Technologies under Grant AHY150300.

References

[1] "Amazon ec2," https://docs.aws.amazon.com/ec2/index.html.

[2] "Alibaba cloud," https://us.alibabacloud.com.

[3] "Amazon virtual private cloud user guide," https://docs.amazonaws.cn/en_us/vpc/latest/userguide/VPC_Subnets.html#vpc-subnet-basics.

[4] S. Zhang, Y. Liu, W. Meng, J. Bu, S. Yang, Y. Sun, D. Pei, J. Xu, Y. Zhang, L. Song *et al.*, "Efficient and robust syslog parsing for network devices in datacenter networks," *IEEE Access*, vol. 8, pp. 30 245–30 261, 2020.

[5] A. O. F. Atya, Z. Qian, S. V. Krishnamurthy, T. La Porta, P. McDaniel, and L. Marvel, "Malicious co-residency on the cloud: Attacks and defense," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[6] A. Arora, S. K. Yadav, and K. Sharma, "Denial-of-service (dos) attack and botnet: Network analysis, research tactics, and mitigation," in *Research Anthology on Combating Denial-of-Service Attacks*. IGI Global, 2021, pp. 49–73.

[7] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: a field study of middlebox failures in datacenters," in *Proceedings of the 2013 conference on Internet measurement*, 2013, pp. 9–22.

[8] C. Tan, Z. Jin, C. Guo, T. Zhang, H. Wu, K. Deng, D. Bi, and D. Xiang, "Netbouncer: Active device and link failure localization in data center networks," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 2019, pp. 599–614.

[9] P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "Intrusion detection techniques in cloud environment: A survey," *Journal of Network and Computer Applications*, vol. 77, pp. 18–47, 2017.

[10] B. Yong, G. Zhang, H. Chen, and Q. Zhou, "Intelligent monitor system based on cloud and convolutional neural networks," *The Journal of Supercomputing*, vol. 73, no. 7, pp. 3260–3276, 2017.

[11] A. Shieh, S. Kandula, A. G. Greenberg, and C. Kim, "Seawall: Performance isolation for cloud datacenter networks." in *HotCloud*, 2010.

[12] 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 *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–12.

[13] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Transactions on Communications*, vol. 64, no. 9, pp. 3746–3758, 2016.

[14] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 50–56.

[15] J. Liu, H. Xu, G. Zhao, C. Qian, X. Fan, and L. Huang, "Incremental server deployment for scalable nfv-enabled networks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2361–2370.

[16] W. JingZhou, Z. Gongming, X. Hongli, H. He, L. Luyao, and Y. Yongqiang, "Robust service mapping in multi-tenant clouds," in *IEEE INFOCOM 2021-The 40th Annual IEEE International Conference on Computer Communications*. IEEE, 2012, pp. 1–11.

[17] 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.

[18] K. Mills, J. Filliben, and C. Dabrowski, "Comparing vm-placement algorithms for on-demand clouds," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*. IEEE, 2011, pp. 91–98.

[19] J. Guo and L. N. Bhuyan, "Load balancing in a cluster-based web server for multimedia applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 11, pp. 1321–1334, 2006.

[20] N. K. Das, M. S. George, and P. Jaya, "Incorporating weighted round robin in honeybee algorithm for enhanced load balancing in cloud environment," in *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2017, pp. 0384–0389.

[21] C. Baun and M. Kunze, "The koala cloud management service: A modern approach for cloud infrastructure management," in *Proceedings of the First International Workshop on Cloud Computing Platforms*, ser. CloudCP '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/1967422.1967423

[22] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, pp. 1–14, 2011.

[23] "Tencent cloud instance types," intl.cloud.tencent.com/document/product/213/11518.