Robust Service Mapping in Multi-Tenant Clouds

Jingzhou Wang[1,3] Gongming Zhao[1,3] Hongli Xu[1,3] He Huang[2] Luyao Luo[1,3] Yongqiang Yang[4]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]School of Computer Science and Technology, Soochow University
[3]Suzhou Institute for Advanced Study, University of Science and Technology of China
[4]Huawei Technologies Co., Ltd

***Abstract*—In a multi-tenant cloud, cloud vendors provide services (*e.g.*, elastic load-balancing, virtual private networks) on service nodes for tenants. Thus, the mapping of tenants' traffic and service nodes is an important issue in multi-tenant clouds. In practice, unreliability of service nodes and uncertainty/dynamics of tenants' traffic are two critical challenges that affect the tenants' QoS. However, previous works often ignore the impact of these two challenges, leading to poor system robustness when encountering system accidents. To bridge the gap, this paper studies the problem of robust service mapping in multi-tenant clouds (RSMP). Due to traffic dynamics, we take a two-step approach: *service node assignment* and *tenant traffic scheduling*. For service node assignment, we prove its NP-Hardness and analyze its problem difficulty. Then, we propose an efficient algorithm with bounded approximation factors based on randomized rounding and knapsack. For tenant traffic scheduling, we design an approximation algorithm based on fully polynomial time approximation scheme (FPTAS). The proposed algorithm achieves the approximation factor of 2+ϵ, where ϵ is an arbitrarily small value. Both small-scale experimental results and large-scale simulation results show the superior performance of our proposed algorithms compared with other alternatives.**

***Index Terms*—Multi-Tenant Cloud, Service Mapping, Load Balancing, Robustness**

I. Introduction

Building IT infrastructure by enterprises/individuals incurs many costs, ranging from hardware costs to software costs (*e.g.*, purchasing hardware, licensing software and managing enterprise networks) [1]. Cloud computing has enabled a new paradigm to host different services on data centers maintained by professional cloud vendors [2], *e.g.*, Amazon Web Services [3] and Google Cloud Platform [4]. Cloud vendors free users from cumbersome tasks (*e.g.*, managing and maintaining IT infrastructure) through centralized management, thus an increasing number of enterprises/individuals are moving workloads to clouds [5] [6].

In a multi-tenant cloud, two or more tenants share the same hardware resources, *e.g.*, creating VMs on the same physical machine (referred to as a *computing node* hereafter) [7]. Tenants may require the support of various network services, *e.g.*, elastic load-balancing (ELB) [8], virtual private networks (VPN) [9], firewall [10] and data storage auditing service [11]. Thus, in the cloud environment, a set of software/hardware devices has been configured for providing various services, referred to as *service nodes* hereafter. According to tenants' requirements, the cloud vendor is responsible for scheduling tenants' traffic to the proper service node(s), also called *service mapping* [12] [13].

To solve the service mapping problem in multi-tenant clouds, several efficient solutions have been designed in [14] [15] [16] [17]. Existing works usually focus on the problem of traffic scheduling with minimum makespan under resource constraints [14] [15] or with minimum cost under deadline constraints [16] [17]. For example, the authors [14] consider service node processing delay and formulate the joint problem of service mapping and traffic steering as a mixed integer linear program with the objective of minimum makespan. The work [16] aims to minimize the workflow execution cost with deadline constraints.

However, it is far from trivial to achieve a robust service mapping scheme in a multi-tenant cloud using existing methods. Specifically, there are two critical but neglected challenges for service mapping in practice. One is *unreliability of service nodes.* Network device failure is a common scenario in today's networks. For example, the previous work [18] has shown that load balancers experience 21% of device failures in one year. Service node failure will cause the service unavailable and decrease tenants' QoS, especially in the commercial clouds [19]. Therefore, we should try to reduce the number of affected tenants when encountering service node failure. The other challenge is *uncertainty of tenants' traffic.* In practice, traffic in a multi-tenant cloud is diverse due to different purposes and expectations of various tenants. On one hand, malicious tenants are very common in multi-tenant clouds [20]. They may launch network attacks [20] [21] and make the service nodes unavailable [22]. For instance, AWS has already been attacked by spammers and often been subject to denial of service attacks [23]. On the other hand, the traffic may change dynamically to meet tenants' requirements [24]. Thus, bursty traffic is common in clouds and may cause the mapped service node(s) overload [25]. For example, the new computing paradigms, *e.g.*, MapReduce [26] and distributed machine learning [27], may cause highly bursty traffic and overload the mapped service node(s) [25]. Thus, it is necessary to control the number of service nodes that a tenant will affect.

To our best knowledge, existing works on service mapping often focus on resource constraints and ignore the unreliability/uncertainty issues in the cloud environment. To conquer the above challenges, this paper studies the problem of robust service mapping in multi-tenant clouds (RSMP). The key idea of RSMP is two-fold: 1) In order to reduce the impact of service node unreliability on tenants, RSMP should ensure that each service node can only serve a limited number of

tenants. In this way, we can control the number of affected tenants when encountering service node failure. 2) In order to relieve the impact of tenant traffic uncertainty on service nodes, RSMP should ensure that the traffic of each tenant will be forwarded to a limited number of service nodes. In this way, we can control the number of affected service nodes when a tenant generates bursty/anomaly traffic. The main contributions of this paper are as follows:

1) We give the problem of robust service mapping in multi-tenant clouds (RSMP). Due to traffic dynamics, we solve this problem by taking a two-step approach: service node assignment and tenant traffic scheduling.
2) For service node assignment, we analyze the complexity of this problem by showing that several existing NP-hard problems are special cases of our problem. We propose a randomized rounding and knapsack based algorithm for this problem, and prove the approximation factor of $O(\log m_s)$, where m_s is the number of service nodes.
3) For tenant traffic scheduling, we present an efficient algorithm based on binary search and combinatorial rounding. Moreover, we design a fully polynomial time approximation scheme (FPTAS) to further reduce the time complexity. We analyze that the proposed algorithm can achieve (2+ϵ)-approximation, where ϵ is an arbitrarily small value.
4) We conduct small-scale tests and large-scale simulations using real-world topologies and datasets [28] [29] to show that the proposed algorithms achieve superior performance compared with the state-of-the-art solutions.

II. Preliminaries

A. Multi-Tenant Cloud Model

A typical multi-tenant cloud consists of three components: a computing node set, a service node set and a management plane. Let $N = \{n_1, n_2, ...n_{m_n}\}$ represent the set of computing nodes, where $m_n = |N|$ is the number of computing nodes. There are several types of service nodes in a multi-tenant cloud, *e.g.*, ELB and VPN. $E = \{e_1, e_2, ..., e_h\}$ denotes the service type set, where $h = |E|$ is the number of service types in the cloud. Let $S^e = \{s_1^e, s_2^e, ..., s_{m_e}^e\}$ denote the service node set with type $e \in E$, where $m_e = |S^e|$ is the number of service nodes of type $e \in E$. Moreover, we use set $S = S^1 \cup S^2 ... \cup S^h$ to denote the whole service node set and the number of service nodes is denoted by m_s. The management plane is responsible for managing the whole network, including service mapping decisions and system accident management in the network.

In practice, a set of tenants rent VMs and buy services from cloud vendors according to their needs. We use $T = \{t_1, t_2, ...t_{m_t}\}$ to denote the set of tenants, where $m_t = |T|$ is the number of tenants. Moreover, we use set E_j to denote the set of service types purchased by tenant $t_j \in T$. Meanwhile, cloud vendors create VMs on computing nodes and provide services on service nodes according to tenants' requirements. Note that, when a tenant sends a command to create a VM, the cloud management plane selects a computing node to start a VM based on the current load of the computing nodes, thus the VM's location is uncertain and a large-scale tenant may have VMs on thousands of computing nodes [30].

B. Problem Statement

In this section, we give the problem statement of robust service mapping in multi-tenant clouds (RSMP). In multi-tenant clouds, considering that different tenants may generate traffic with various service requirements on different computing nodes, we usually schedule traffic at the granularity of the combination of tenants, computing nodes and required services. In other words, we identify a *request* by three elements (tenant, computing node and service type). For each request, we need to forward it to a proper service node, also called service mapping.

To enhance the system robustness and improve the tenants' QoS, we should consider the following two robustness constraints when mapping tenants' traffic with service nodes. 1) *Tenant Constraint*. Considering the bursty/anomaly traffic, we do not expect that a single tenant's bursty/anomaly traffic affects too many service nodes. Thus, the number of service nodes of each type that a tenant will map to should not exceed k, where k is a constant determined by system requirements. 2) *Service Node Constraint*. We do not expect a single service node failure affects too many tenants. So, a service node is required to serve no more than q tenants, where q is a constant determined by system requirements. Our objective is to achieve load balancing among all service nodes, which can improve service availability and execution efficiency [31].

C. Algorithm Workflow

Tenants may create/delete VMs dynamically and the traffic volume from each VM at different times may vary greatly [32]. In order to adapt to traffic uncertainty/dynamics, we should update service mapping frequently. Service nodes (*e.g.*, IDS and Proxy) often need to record the state of processed traffic [33]. However, frequent mapping updates will increase extra delay and overhead to maintain the flow state consistency on service nodes [33]. For example, after network updates, if a tenant's traffic is scheduled to a service node that does not maintain the traffic's state, it will lead to flow state inconsistency. Some previous works [34] manage to migrate the flow states from the original service nodes to the target ones. However, the state migration will increase network latency. For example, a loss-free move involving state for 500 flows takes 215ms [34], which is unacceptable for many applications. The situation will become more serious if many flows are migrated between service nodes.

To this end, similar to [35], we take a two-step approach: service node assignment and tenant traffic scheduling. The first step is performed at a long-term interval (*e.g.*, one day), and the second step is triggered by accidents, *e.g.*, service node congestion. Specifically, the first step will select a feasible service node set for each tenant with the robustness constraint in Section III. We only need to maintain flow state consistency for the tenant's traffic between these mapped service nodes, decreasing the state maintenance overhead. For tenant traffic scheduling, we will design a fully polynomial

time approximation scheme (FPTAS) for traffic scheduling dynamically in Section IV.

Note that, in the cloud, the type of service required by each request is fixed and each service node usually supports only one specific service [36]. Thus, different types of service nodes are independent when we map traffic to service nodes. For ease of description, we only consider one type of service node in this paper. If the traffic requires multiple services, we only need to classify the traffic according to required service types, and run our proposed algorithms for each service type.

III. Service Node Assignment

This section first defines the problem of service node assignment (SNA) and then analyzes its difficulty. Finally, an approximation algorithm is proposed to solve this problem.

A. Problem Definition for SNA

To adapt to the traffic uncertainty/dynamics, we take a two-step approach for RSMP. The first step solves the SNA problem. Specifically, we select a feasible service node set for each tenant so as to satisfy robustness constraints at a long-term interval. Let b_j represent the total traffic demand of tenant t_j. We use $y_j^p \in \{0,1\}$ to denote whether tenant t_j selects service node s_p or not. Let x_j^p denote the traffic proportion of tenant t_j served by service node s_p. Besides, we use C_p to denote the capacity of service node $s_p \in S$. The SNA problem can be formulated as follows:

$$\min \quad \zeta$$

$$S.t. \begin{cases} \sum_{s_p \in S} x_j^p = 1, & \forall t_j \in T \\ x_j^p \le y_j^p, & \forall t_j \in T, s_p \in S \\ \sum_{s_p \in S} y_j^p \le k, & \forall t_j \in T \\ \sum_{t_j \in T} y_j^p \le q, & \forall s_p \in S \\ \sum_{t_j \in T} x_j^p \cdot b_j \le \zeta \cdot C_p, & \forall s_p \in S \\ x_j^p \in [0,1] & \forall t_j \in T, s_p \in S \\ y_j^p \in \{0,1\}, & \forall t_j \in T, s_p \in S \end{cases} \tag{1}$$

The first set of equations means that all traffic of each tenant will be processed by service nodes. The second set of inequalities denotes whether some traffic of tenant t_j will be served by service node s_p or not. $y_j^p = 1$ means service node s_p will process traffic of tenant t_j. The third set of inequalities denotes the tenant constraint, that is, each tenant's traffic will be processed by at most k service nodes. The fourth set of inequalities represents the service node constraint, that is, each service node can process traffic from at most q tenants. The fifth set of inequalities describes the traffic load on each service node, where $\zeta \in [0,1]$ represents the load-balancing factor. Our objective is to achieve load balancing on all service nodes, *i.e.*, min ζ.

B. Problem Complexity Analysis

By ignoring the tenant constraint (*i.e.*, $k = m_s$) and the service node constraint (*i.e.*, $q = m_t$), the SNA problem becomes a typical parallel machines scheduling (PMS) problem [37]. As a result, the SNA problem is NP-Hard too. In fact, the problem remains challenging even relaxing any one of the two constraints. The simplified problem includes special cases like routing and distribution problems, which show the complexity of our problem.

Definition 1 (K-Splittable Routing (KSR) problem [35]): There is a network topology including a set of flows $\gamma = \{r_1, r_2, ..., r_{|\gamma|}\}$ each associated with a traffic size $f(r_i)$. We determine a set of potential paths P_i for each flow r_i. We will choose at most k paths in P_i for flow r_i so as to minimize the maximum load factor of all links.

According to [35], there exists a complex rounding-based algorithm with the approximation factor of $O(\log N)$, where N denotes the number of links in the topology.

Difference from the KSR problem: In this case, we regard each tenant as a flow with traffic size b_i, and choose at most k paths/service nodes for each flow/tenant. If we ignore the service node constraint, SNA becomes the KSR problem.

Definition 2 (Data Distribution (DD) Problem [38]): There are a set of M servers and a set of N documents. Each server is associated with a memory size m_i. Each document j is associated with a document size s_j and an access cost r_j. We need to choose servers for each document with the server's memory size constraint. The objective is to minimize the maximum access cost among all servers.

According to [38], there exists an algorithm which can achieve the optimal solution, violating the access cost by at most a factor $2(1 + \frac{1}{\psi})$ and the memory size by at most a factor $2(1 + \frac{1}{\psi})$, where ψ denotes the maximum number of documents saved by a server.

Difference from the DD problem: We regard each tenant as a document with one unit document size and access cost b_i. A service node is actually a server which can only accept q documents. If a tenant can map to unlimited service nodes, *i.e.*, ignoring the tenant constraint, we say that the DD problem is a special case of our SNA problem.

The above analysis shows that the SNA problem is much more difficult than both KSP and DD. Thus, designing an algorithm with bounded approximation factors to solve the SNA problem is far from trivial and in urgent need.

C. Algorithm Design for SNA

Similar to [39], we can adopt the random rounding method to solve the SNA problem. The approximation factors for this solution are $(O(\log m_s), O(\log m_s), O(\log m_s))$, which represent the maximum exceeded factors of tenant constraint, service node constraint and load balancing factor constraint, respectively. To improve the approximation performance, we present an approximation algorithm, called Rounding-based Service Node Assignment (RSNA), to solve this problem with approximation factors of $(1, O(\log m_s), O(\log m_s))$, that is, our algorithm can strictly satisfy the tenant constraint. RSNA mainly consists of two steps. The first step relaxes Eq. (1) by replacing the seventh line of integer constraints with $y_j^p \in [0,1]$, turning the problem into linear programming. We can solve it with a linear program solver (*e.g.*, CPLEX [40]) and the solutions are denoted as $\{\widetilde{x}_j^p\}$, $\{\widetilde{y}_j^p\}$ and $\widetilde{\zeta}$.

In the second step, we determine how to assign service nodes for each tenant, *i.e.*, obtain feasible solutions $\{\widehat{y}_j^p\}$

and $\{\widehat{x}_j^p\}$. For each tenant t_j, we first define another integer variable $k(j) = \lfloor \sum_{s_p \in S} \widetilde{y}_j^p \rfloor$. Then we put all variables $\widetilde{y}_j^p$ $(\forall s_p \in S)$ into $k(j)$ knapsacks so as to minimize the sum of all variables in each knapsack. For each knapsack a, assume that it contains a set of variables, denoted as V_a, and let $z_a = \sum_{\widetilde{y}_j^p \in V_a} \widetilde{y}_j^p$. One variable $\widetilde{y}_j^p$ will be chosen (*i.e.*, $\widehat{y}_j^p = 1$) with probability $\frac{\widetilde{y}_j^p}{z_a}$, and the traffic proportion of tenant t_j served by service node s_p is $\widehat{x}_j^p = \frac{\widetilde{x}_j^p \cdot z_a}{\widetilde{y}_j^p}$.

Note that, after we put variables into knapsacks with the objective of min-max sum, z_a is usually approximately equal to 1 and we will prove that z_a must be greater than 0.5. According to the first and third constraints in Eq. (1), we know that $\widetilde{x}_j^p$ is generally much smaller than $\widetilde{y}_j^p$. Thus we believe $\widehat{x}_j^p \leq 1$. For each tenant t_j, we select the set of service nodes with $\widehat{y}_j^p = 1$ as the feasible service node set. The RSNA algorithm is formally described in Algorithm 1.

Algorithm 1 Rounding-based Service Node Assignment

1: **Step 1: Solving the Relaxed SNA Problem**
2: Construct a linear program by replacing with $y_j^p \in [0, 1]$
3: Obtain the optimal solutions $\{\widetilde{x}_j^p\}$ and $\{\widetilde{y}_j^p\}$
4: **Step 2: Service Node Assignment for Each Tenant**
5: **for** each tenant $t_j \in T$ **do**
6: Let $k(j) = \lfloor \sum_{s_p \in S} \widetilde{y}_j^p \rfloor$
7: Put all variables $\widetilde{y}_j^p$ $(\forall s_p \in S)$ into $k(j)$ knapsacks with min-max sum
8: **for** each knapsack a **do**
9: Let set V_a denote the variables in knapsack a
10: $z_a = \sum_{\widetilde{y}_j^p \in V_a} \widetilde{y}_j^p$
11: Choose s_p for $\widetilde{y}_j^p \in V_a$ with probability $\frac{\widetilde{y}_j^p}{z_a}$
12: Let $\widehat{x}_j^p = \frac{\widetilde{x}_j^p \cdot z_a}{\widetilde{y}_j^p}$ for the chosen service node s_p
13: Select the service nodes with $\widehat{y}_j^p = 1$ for tenant t_j

D. Performance Analysis

This section proves the correctness of our RSNA algorithm and analyzes its approximate performance.

Theorem 1: The proposed RSNA algorithm guarantees that each tenant $t_j \in T$ will choose at most k service nodes, *i.e.*, we can strictly guarantee the tenant constraint.

Proof: For each tenant $t_j \in T$, we put all service nodes into $k(j)$ knapsacks and select a service node in one knapsack, thus we will choose $k(j)$ service nodes in total. On the other hand, according to the definition of variable $k(j)$ and the third constraints in Eq. (1) we have :

$$k(j) = \lfloor \sum\nolimits_{s_p \in S} \widetilde{y}_j^p \rfloor \leq \sum\nolimits_{s_p \in S} \widetilde{y}_j^p \leq k \tag{2}$$

Thus, we can strictly guarantee the tenant constraint. ■

Lemma 2: The lower bound of the sum z_a of all variables in any knapsack a for each tenant t_j is greater than $\frac{1}{2}$ and not greater than $\frac{k(j)}{2k(j)-1}$.

Proof: We first prove that the lower bound is greater than $\frac{1}{2}$. For each tenant t_j, according to the definition of $k(j)$, we have:

$$\sum\nolimits_{s_p \in S} \widetilde{y}_j^p = k(j) + \epsilon,\ 0 < \epsilon < 1 \tag{3}$$

We define two sets:

$$\begin{cases} Y_1 = \{\widetilde{y}_j^p | \frac{1}{2} < \widetilde{y}_j^p < 1, s_p \in S\} \\ Y_2 = \{\widetilde{y}_j^p | 0 < \widetilde{y}_j^p < \frac{1}{2}, s_p \in S\} \end{cases} \tag{4}$$

We arbitrarily choose two numbers from set Y_2 (*e.g.*, y_p^1 and y_p^2) and compute their sum (*e.g.*, $y_p = y_p^1 + y_p^2$). If the result is less than 0.5, we put the new value into Y_2 and delete the chosen two numbers from Y_2 (*e.g.*, $Y_2 = Y_2 - \{y_p^1, y_p^1\} + \{y_p'\}$). Otherwise, we put it into set Y_1 and delete the chosen two numbers from Y_2. (*e.g.*, $Y_1 = Y_1 + \{y_p'\}$, $Y_2 = Y_2 - \{y_p^1, y_p^1\}$). We repeat the above operations until there is at most one number in Y_2. Now, we assume there are less than $k(j)$ numbers in Y_1, *i.e.*, there are at most $k(j) - 1$ numbers in Y_1. Note that the numbers in Y_1 are all less than 1. Then

$$\sum\nolimits_{\widetilde{y}_j^p \in Y_1} \widetilde{y}_j^p < k(j) - 1 \tag{5}$$

The sum of the remaining numbers in Y_2 is

$$\sum\nolimits_{\widetilde{y}_j^p \in Y_2} \widetilde{y}_j^p < \frac{1}{2} \tag{6}$$

Thus, we have

$$\sum\nolimits_{s_p \in S} \widetilde{y}_j^p < k(j) - \frac{1}{2} \tag{7}$$

However, it contradicts $\sum_{s_p \in S} \widetilde{y}_j^p = k(j) + \epsilon$ in Eq. (3). Thus, there are at least $k(j)$ numbers in Y_1. In this case, we can simply assign each $\widetilde{y}_j^p$ in Y_1 to each knapsack. Then the total value z_a in any knapsack a must be greater than $\frac{1}{2}$.

We next prove that the lower bound is not greater than $\frac{k(j)}{2k(j)-1}$. We assume that there are $2k(j) - 1$ identical numbers, each with a value of $\frac{k(j)}{2k(j)-1}$. When we assign these numbers to $k(j)$ knapsacks, it is obvious that there exists a knapsack which only has one number with the value of $\frac{k(j)}{2k(j)-1}$. Thus, there does not exist an assignment scheme to make sure the sum of variables in any knapsack is greater than $\frac{k(j)}{2k(j)-1}$. ■

Theorem 3: The proposed RSNA algorithm guarantees that the number of tenants on any service node will not exceed the service node constraint by a factor of $O(\log m_s)$, where m_s is the number of service nodes.

Proof: We first prove that for each tenant $t_j \in T$ and service node $s_p \in S$, we have $\mathbb{E}\left[\widehat{y}_j^p\right] \leq 2 \cdot \widetilde{y}_j^p$. Specifically, for any variable $\widetilde{y}_j^p$ in knapsack a, we round $\widetilde{y}_j^p$ to 1 with probability $\frac{\widetilde{y}_j^p}{z_a}$. Thus, we have $\mathbb{E}\left[\widehat{y}_j^p\right] = \frac{\widetilde{y}_j^p}{z_a}$. According to Lemma 2, we have $\mathbb{E}\left[\widehat{y}_j^p\right] = \frac{\widetilde{y}_j^p}{z_a} \leq 2 \cdot \widetilde{y}_j^p$. Then, we analyze the approximation ratio performance based on the classical randomized rounding method. Due to space limit, we omit the proof. The reader can refer to [41] [42] for the performance analysis of the randomized rounding method. ■

Theorem 4: The RSNA algorithm can achieve the approximation factor of $O(\log m_s)$ for load balancing factor in multi-tenant clouds, where m_s is the number of service nodes.

Proof: We first prove that for each tenant $t_j \in T$ and service node $s_p \in S$, we have $\mathbb{E}\left[\widehat{x}_j^p\right] = \widetilde{x}_j^p$. Specifically,

for any variable $\widetilde{x}_j^p$ in knapsack a, we let $\widehat{x}_j^p = \frac{\widetilde{x}_j^p \cdot z_a}{\widetilde{y}_j^p}$ with probability $\frac{\widetilde{y}_j^p}{z_a}$. Thus, we have $\mathbb{E}\left[\widehat{x}_j^p\right] = \frac{\widetilde{y}_j^p}{z_a} \cdot \frac{\widetilde{x}_j^p \cdot z_a}{\widetilde{y}_j^p} = \widetilde{x}_j^p$. Similarly, we can analyze approximation ratio performance based on the classical randomized rounding method. The reader can refer to [41] [42] for the performance analysis of the randomized rounding method. ■

IV. Tenant Traffic Scheduling

In this section, we first define the problem of tenant traffic scheduling (TFS). We then propose an approximation algorithm, called binary search and combinatorial rounding based traffic scheduling (BCTS). Finally, we analyze its approximation performance and time complexity.

A. Problem Definition for TFS

By the RSNA algorithm as described in Section III, we use S_j to denote the set of available service nodes for each tenant t_j. Let $x_{j,n}^p \in \{0,1\}$ denote whether the traffic of tenant $t_j \in T$ on computing node $n \in N$ maps to service node $s_p \in S$ or not. We use $b_{j,n}$ to represent the total traffic demand of tenant $t_j \in T$ on computing node $n \in N$. Accordingly, we formulate the TFS problem as follows:

$$\min \quad \lambda$$
$$S.t. \begin{cases} \sum_{s_p \in S_j} x_{j,n}^p = 1, & \forall t_j \in T, n \in N \\ \sum_{t_j \in T, n \in N} x_{j,n}^p b_{j,n} \le \lambda \cdot C_p, & \forall s_p \in S_j \\ x_{j,n}^p \in \{0,1\}, & \forall t_j, n, s_p \end{cases} \tag{8}$$

The first set of equations denotes that the traffic of a tenant on one computing node will be assigned to one service node. The second set of inequalities states the traffic load on service node s_p, where the λ is called as the load ratio. Our goal is to achieve the load balancing among all service nodes, that is, min λ.

B. Algorithm Design for TFS

The problem in Eq. (8) can be regarded as an unrelated parallel machine scheduling (UPMS) problem [43]. If we solve this problem using classical algorithms [43], considering the time complexity of these algorithms, it may not achieve good results in dynamic clouds. Thus, this section leverages the FPTAS method to design an efficient approximation algorithm, called binary search and combinatorial rounding based traffic scheduling (BCTS). BCTS is formally described in Algorithm 2.

The algorithm consists of three main steps. The first step of BCTS is to relax TFS into a linear program, in which $x_{j,n}^p$ can be fractional. According to the above relaxation, the TFS problem is transformed as follows:

$$\min \quad \lambda$$
$$S.t. \begin{cases} \sum_{s_p \in S_j} x_{j,n}^p = 1, & \forall t_j \in T, n \in N \\ \sum_{t_j \in T, n \in N} x_{j,n}^p b_{j,n} \le \lambda \cdot C_p, & \forall s_p \in S_j \\ x_{j,n}^p \in [0,1] \end{cases} \tag{9}$$

Then the second step is to derive a fractional solution of Eq. (9) using binary search and fully polynomial time approximation scheme (FPTAS). According to problem definition in Eq. (8), we know the upper bound of λ is $\Theta_h = \frac{\sum_{t_j \in T, n \in N} b_{j,n}}{\min_{s_p \in S}(C_p)}$, that is, all traffic is processed on the service node with the minimum capacity. Besides, the lower bound of λ is $\Theta_l = \frac{\max_{t_j \in T, n \in N}(b_{j,n})}{\max_{s_p \in S}(C_p)}$, that is, the request with the maximum traffic demand is processed on the service node with the maximum capacity. We next use binary search to obtain the feasible solutions. Specifically, let $\Theta_m = \frac{\Theta_l + \Theta_h}{2}$. We first remove some inefficient assignments between requests and service nodes that $\frac{b_{j,n}}{C_p} > \Theta_m$, *i.e.*, let $x_{j,n}^p = 0$ if $\frac{b_{j,n}}{C_p} > \Theta_m$. After removing those assignments, if we find Eq. (9) is not solvable, we will search in the upper range, *i.e.*, let $\Theta_l = \Theta_m$. Otherwise, we will search in the lower range, *i.e.*, let $\Theta_h = \Theta_m$. Clearly, when Θ_m approaches Θ_h, Eq. (9) becomes the ordinary LP, which must be solvable. We repeat the binary search operation until $|\Theta_h - \Theta_l| \le \xi$, where ξ represents the search accuracy.

For ease of description, let I denote the request set and the traffic demand of request $i \in I$ is denoted as b_i. Let Φ denote the set of all feasible assignments in the cloud and Φ_i denote the set of feasible assignments of request i. We use $\phi_{i,p} \in \Phi$ to denote that request i is assigned to service node s_p and the traffic amount of $\phi_{i,p}$ is denoted as $w_{i,p}$. The TFS problem can be transformed as follows:

$$\min \quad \eta$$
$$S.t. \begin{cases} \sum_{\phi_{i,p} \in \Phi} w_{i,p} \le \eta \cdot C_p, & \forall s_p \in S \\ \sum_{\phi_{i,p} \in \Phi_i} w_{i,p} \ge b_i, & \forall i \in I \\ w_{i,p} \ge 0, & \forall \phi_{i,p} \in \Phi \end{cases} \tag{10}$$

The first set of inequalities denotes the total traffic load on a service node. The second set of inequalities means that all traffic of each request will be processed. Then we give an equivalent form of Eq. (10) as follows:

$$\max \quad \eta'$$
$$S.t. \begin{cases} \sum_{\phi_{i,p} \in \Phi} w_{i,p} \le C_p, & \forall s_p \in S \\ \sum_{\phi_{i,p} \in \Phi_i} w_{i,p} \ge \eta' \cdot b_i, & \forall i \in I \\ w_{i,p} \ge 0, & \forall \phi_{i,p} \in \Phi \end{cases} \tag{11}$$

Compared with Eq. (10), we find that the optimal solution of Eq. (11) is the inverse of that of Eq. (10). Furthermore, we give the dual problem of Eq. (11) as follows:

$$\min \quad \sum_{s_p \in S} C_p l_p$$
$$S.t. \begin{cases} l_p \ge r_i, & \forall \phi_{i,p} \in \Phi_i, i \in I \\ \sum_{i \in I} b_i r_i \ge 1 \\ l_p \ge 0, & \forall s_p \in S \end{cases} \tag{12}$$

Note that the dual variables l_p and r_i represent the service node capacity constraints and request traffic demand constraints in Eq. (11), respectively. Assume that L_i represents the service node with the least l_p mapped to request i. Then, Eq. (12) can transform into Eq. (13).

Algorithm 2 BCTS: Binary Search and Combinatorial Rounding based Traffic Scheduling

Step 1: Relaxing the TFS Problem
Construct a linear program in Eq. (9)
Step 2: Deriving a Fractional Solution using FPTAS
Initialize Θ_l, Θ_h
while $|\Theta_h - \Theta_l| > \xi$ **do**
  $\Theta_m = \frac{\Theta_l + \Theta_h}{2}$
  Compute Φ_i for each b_i according to Θ_m
  Transform Eq. (9) into Eq. (13)
  Apply the FPTAS method to solve Eq. (13) and get the fractional solution λ_F
  if The solution λ_F does not exist or $\lambda_F > \Theta_m$ **then**
    $\Theta_l = \Theta_m$
  else
    $\Theta_h = \Theta_m$
Step 3: Generating an Integral Solution
Construct a bipartite graph
Use the Hungarian algorithm to find a matching $\mathbb{A}$ covering all requests
Return the final assignment based on the matching $\mathbb{A}$

$$\min \sum_{s_p \in S} C_p l_p$$
$$S.t. \begin{cases} \sum_{i \in I} b_i L_i \geq 1 \\ l_p \geq 0, \qquad \forall s_p \in S \end{cases} \tag{13}$$

Similar to the work [44], we can apply FPTAS with low time complexity to obtain a fractional solution for Eq. (13).

After we derive the fractional solution λ_F, the last step is to construct a feasible integer solution using combinatorial rounding. Based on the fractional solution, we first construct a bipartite graph, which consists of two sets of vertices: a set of request vertices and a set of service node vertices. For each $0 < x^p_{j,n} < 1$, there is an edge between the request vertex $b_{j,n}$ and the service node vertex s_p. For this bipartite graph, we use the Hungarian algorithm [45] to find a matching $\mathbb{A}$ that covers all requests, and the matching $\mathbb{A}$ can acquire λ_B and the final assignment scheme by assigning each request to the corresponding service node.

C. Performance Analysis

Lemma 5: The fractional solution λ_F by FPTAS is no more than $(1+\epsilon) \cdot \lambda_{lp}$, where ϵ is a value between $[0, 1]$, and λ_{lp} denotes the optimal fractional solution of Eq. (9).

Lemma 6: The solution λ_B by combinatorial rounding process is no more than $\lambda_F + \lambda_{ip}$, where λ_{ip} denotes the optimal integer solution of Eq. (8).

Due to space limit, we omit the proofs of lemmas 5 and 6. The readers can refer to the analysis of FPTAS [46] and combinatorial rounding method [43] for the detailed proofs.

Theorem 7: The approximate factor of BCTS is $2 + \epsilon$.

Proof: According to the above two lemmas, first, the solution λ_F by FPTAS is no more than $(1+\epsilon) \cdot \lambda_{lp}$. Second, using the combinatorial rounding process, we can get a solution λ_B with no more than $\lambda_F + \lambda_{ip}$, where λ_{ip} denotes the optimal integer solution of Eq. (8). Then we have

$$\begin{aligned} \lambda_B &\leq \lambda_F + \lambda_{ip} \leq (1+\epsilon) \cdot \lambda_{lp} + \lambda_{ip} \\ &\leq (1+\epsilon) \cdot \lambda_{ip} + \lambda_{ip} = (2+\epsilon) \cdot \lambda_{ip} \end{aligned} \tag{14}$$

The third inequality holds because the optimal fractional solution should not exceed the optimal integer solution. Based on the above analysis, we can conclude that the BCTS algorithm guarantees the approximation factor of $2 + \epsilon$. ■

Theorem 8: The time complexity of BCTS is $O(\frac{\mathbb{M}^2}{\epsilon^2} \log \frac{\Theta}{\xi} \log^{O(1)} \mathbb{M} + |\mathbb{V}| \cdot |\mathbb{E}|)$, where $\mathbb{M}$ is the number of feasible schemes and Θ equals $\frac{\sum_{t_j \in T, n \in N} b_{j,n}}{\min_{s_p \in S}(C_p)}$. Let $\mathbb{V}$ and $\mathbb{E}$ represent the set of vertices and edges, respectively, in the bipartite graph.

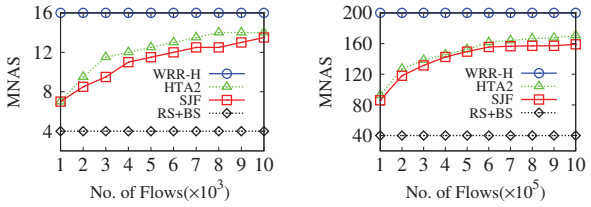
Proof: The time complexity of BCTS mainly consists of three parts: binary search, FPTAS and combinatorial rounding. The running time of binary search, FPTAS, and combinatorial rounding is $O(\frac{\Theta}{\xi})$ [47], $O(\frac{\mathbb{M}^2}{\epsilon^2} \log^{O(1)} \mathbb{M})$ [44] and $O(|\mathbb{V}| \cdot |\mathbb{E}|)$ [45], respectively. Thus, the total running time of BCTS is $O(\frac{\mathbb{M}^2}{\epsilon^2} \log \frac{\Theta}{\xi} \log^{O(1)} \mathbb{M} + |\mathbb{V}| \cdot |\mathbb{E}|)$, which is much lower than the running time by using the standard linear program solvers such as CPLEX [40]. ■

V. Performance Evaluation

A. Performance Metrics and Benchmarks

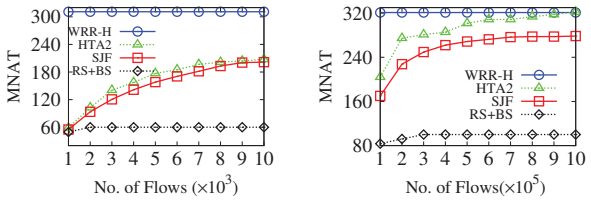
Performance Metrics: We adopt the following performance metrics in evaluation: (1) the maximum number of affected service nodes (MNAS); (2) the maximum number of affected tenants (MNAT); (3) the system throughput; (4) the load balancing factor. During a simulation run, we measure the number of service nodes that each tenant maps to and record the largest value as MNAS. Similarly, we measure the number of tenants served by each service node and record the largest value as MNAT. We calculate the total load of all the service nodes as the system throughput. Moreover, we measure the load of each service node and record the maximum load of all service nodes. For each service node, we divide the load of this service node by its processing capacity to get the load factor. The load balancing factor is the maximum load factor among all service nodes. For simplicity, we use SN to denote service nodes.

Benchmarks: This paper divides RSMP into two subproblems: SNA and TFS. Accordingly, we propose the RSNA algorithm and the BSTC algorithm for SNA and TFS, respectively. The combined algorithm for RSMP is denoted as RS+BS for simplicity. We compare RS+BS with three benchmarks. The first benchmark is the Shortest Job First (SJF) algorithm [48], which is widely adopted in clouds. SJF always chooses the service node with the least burden for the request with the least traffic demand. The second benchmark is the Weighted Round Robin in Honeybee (WRR-H) algorithm [49]. WRR-H uses the Honeybee Inspired algorithm by assigning weights to each service node, and the request is scheduled to the service node according to the traffic demand. The third benchmark is the Heterogeneous Task Assignment algorithm (HTA2) [50]. HTA2 first sorts all the tenants by their traffic demand in the descending order. Then for each



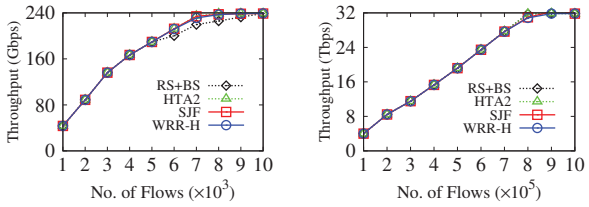
(a) the small topology (b) the large topology

Fig. 1: MNAS vs. Number of Flows without Robustness Constraints



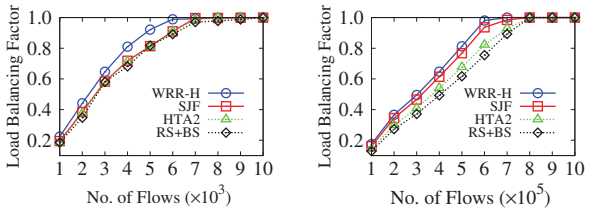
(a) the small topology (b) the large topology

Fig. 2: MNAT vs. Number of Flows without Robustness Constraints



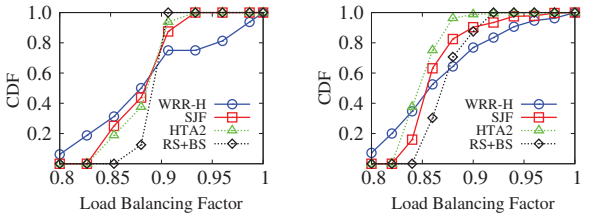
(a) the small topology (b) the large topology

Fig. 3: System Throughput vs. Number of Flows without Robustness Constraints



(a) the small topology (b) the large topology

Fig. 4: Load Balancing Factor vs. Number of Flows without Robustness Constraints



(a) the small topology (b) the large topology

Fig. 5: CDF vs. Load Balancing Factor without Robustness Constraints

tenant, it assigns the request with minimum traffic demand to the service node with the least burden.

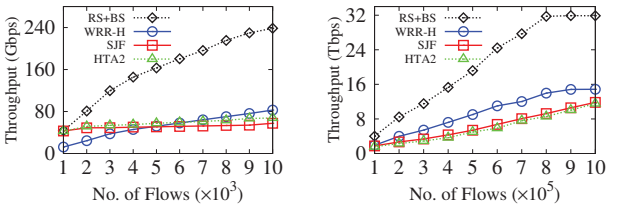
The simulations are performed under two scenarios. The first scenario does not consider robustness constraints for the three benchmarks. This scenario mainly tests MNAS if a tenant sends malicious traffic, and MNAT if a service node encounters a failure. The second scenario considers robustness constraints. In order to guarantee the system robustness, we limit the number of service nodes that each tenant can map to and the number of tenants that each service node can serve for these benchmarks. This scenario mainly tests the system throughput for different algorithms.

B. Simulation Evaluation

1) ***Simulation Settings****:* We conduct simulation experiments to compare RS+BS with three benchmarks in two practical topologies. The first one is a small-scale NSF network topology, which contains 16 service nodes [28]. Since the topology does not provide computing nodes and tenant information, we leverage the ratio of switches and computing nodes (1:30) demonstrated in the Google cluster-data [29] to set the number of computing nodes as 480. Moreover, the number of tenants is set to be the same as the computing nodes. We mainly consider the individual tenants in this topology and let each tenant create 1-20 VMs. The second topology is from Google cluster-data [29], which contains 10047 computing nodes and 324 service nodes. The number of tenants is set to 1000. Moreover, we mainly consider the enterprise tenants in this large-scale topology and let each tenant create 1-200 VMs. We generate tenants' traffic (flows) according to the Google cluster-data [29] and randomly select a VM for each flow. Besides, we generate 6×10^3 flows for the small topology and generate 6×10^5 flows for the large topology by default. k is set $4/40$ and q is set $60/100$ for the small/large topology by default.

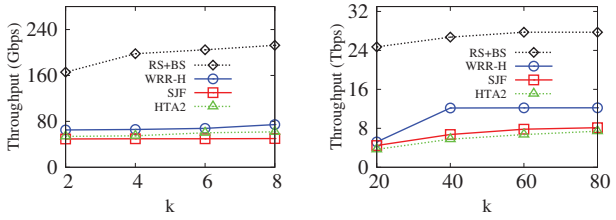
2) ***Performance Comparison without Robustness Constraints****:* The results are shown in Figs. 1-5. Fig. 1 shows that as the number of flows increases, our proposed algorithm always acquires the least MNAS compared with other algorithms. For example, when there are 6×10^3 flows in the small topology, the MNAS results of four algorithms are 11, 16, 12.5 and 4 corresponding to SJF, WRR-H, HTA2 and RS+BS. RS+BS reduces MNAS by 63.6%, 75% and 68% compared with SJF, WRR-H and HTA2, respectively. In comparison, MNAS increases significantly by both SJF and HTA2 since these two algorithms do not limit the number of service nodes a tenant can map to. Besides, WRR-H always acquires the worst result since it adopts the scheme by assigning each request to service nodes in turn. As shown in Fig. 2, we can see that as the number of flows increases, RS+BS always acquires the least MNAT compared with other benchmarks. For example, in Fig. 2(b), when there are 5×10^5 flows, RS+BS can reduce MNAT by 62.7%, 68.8% and 66.8% compared with SJF, WRR-H and HTA2, respectively.

Figs. 3-5 show the throughput and load-balancing performance among all algorithms. Specifically, Fig. 3 shows



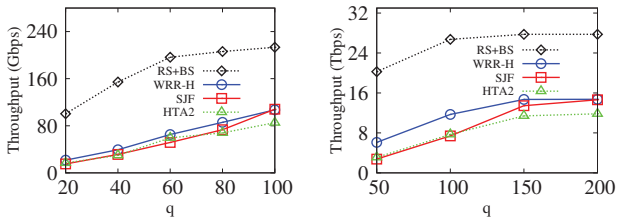
(a) the small topology (b) the large topology

Fig. 6: System Throughput vs. Number of Flows with Robustness Constraints



(a) the small topology (b) the large topology

Fig. 7: System Throughput vs. k with Robustness Constraints



(a) the small topology (b) the large topology

Fig. 8: System Throughput vs. q with Robustness Constraints

that, with the increasing of flows, the system throughput of all algorithms increases in both small and large topologies. RS+BS achieves similar performance compared with the other three benchmarks. Fig. 4 shows that the four algorithms can achieve a similar load balancing factor. We should note that RS+BS achieves similar throughput performance compared with other benchmarks while taking robustness constraints into consideration as shown in Figs. 1-2. Fig. 5 shows the CDF of load balancing factor of these algorithms and RS+BS has the steepest function curve. For instance, in Fig. 5(a), the ratio of service nodes with load balancing factor below 90% is 87.5%, 75%, 93.7% and 100% corresponding to SJF, WRR-H, HTA2 and RS+BS, respectively.

From Figs. 1-5, we can draw a conclusion that without robustness constraints, the three benchmarks will affect more service nodes when a tenant sends malicious traffic to mapped service nodes and will affect more tenants when a service node failure takes place. It may cause network unreliability when dealing with service node failure and tenants' abnormal traffic. Meanwhile, RS+BS can almost achieve the same load-balancing performance compared with other benchmarks.

*3) **Performance Comparison with Robustness Constraints**:* By default, we set $k = 4, q = 60$ in the small topology and set $k = 40, q = 100$ in the large topology. When we assign a request to a service node and find this will violate the robustness constraints, this service node will refuse to serve this request. As shown in Fig. 6(a), when there are 10×10^3 flows in the small topology, RS+BS improves the throughput by 314%, 188% and 252% compared with HTA2, WRR-H and SJF, respectively.

By Fig. 6(b), when there are 10×10^5 flows in the large topology, RS+BS improves the throughput by 169%, 69.8%, 176% compared with SJF, WRR-H and HTA2, respectively.

As shown in Figs. 7-8, when k and q increase, the throughput of the other three benchmarks all grows slowly while that of RS+BS grows fast. For example, in Fig. 7(a), when $k = 8$, RS+BS improves the throughput by 370%, 216% and 281% compared with SJF, WRR-H and HTA2, respectively. In Fig. 8(b), when $q = 150$, RS+BS improves the throughput by 92.0%, 88.6% and 143% compared with SJF, WRR-H and HTA2, respectively. The reason is that other three benchmarks will abandon much traffic to satisfy robustness constraints while RS+BS can assign as much traffic as possible to service nodes.

From these simulation results, we can draw some conclusions. First, from Figs. 1-2, RS+BS can significantly reduce the maximum number of affected service nodes by 63.6%, 75%, 68% on average compared with SJF, WRR-H, and HTA2, respectively, in the large topology. In the small topology, RS+BS can reduce the maximum number of affected service nodes by 74.2%, 80%, 75.2% and reduce the maximum number of affected tenants by 63.3%, 68.8%, 66.1% compared with SJF, WRR-H, HTA2, respectively. Second, Figs. 3-5 show that RS+BS can achieve similar throughput of all service nodes and load balancing result while the other three algorithms do not consider the robustness constraints. Finally, as shown in Figs. 6-8, RS+BS improves the system throughput by about 314%, 188% and 252% on average in the small topology and 169%, 69.8% and 176% in the large topology compared with SJF, WRR-H and HTA2, respectively.

C. Testbed Evaluation

This section presents our system implementation to evaluate these algorithms in a small-scale testbed.

*1) **System Implementation**:* In general, we use 10 servers to build a small-scale testbed. Specifically, we first run SNAT (Static Network Address Translation) service on 5 physical servers all with Ubuntu 16.04-server OS, a core i5-7500 processor and 8G of RAM. Then we adopt 5 physical servers all with a core i5-6500 processor and 8G of RAM as computing nodes which contain tenants' VMs. We assume there are 20 tenants and each tenant has 1-5 VMs. In the testbed, we set $k = 2$, $q = 10$ and 60 flows by default.

We implement our tests with a set of flows from Google cluster-data [29] and the number of generated flows ranges from 20 to 100. We use iPerf3 tool [51] to generate tenants' traffic and send it to the corresponding service nodes. More-

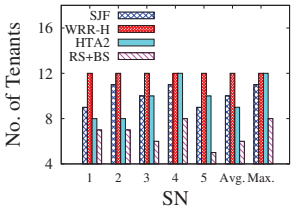


Fig. 9: No. of Tenants of each SN without Robustness Constraints

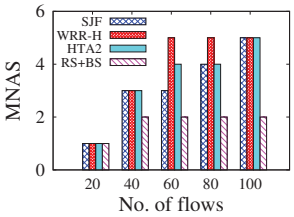


Fig. 10: MNAS vs. No. of Flows without Robustness Constraints

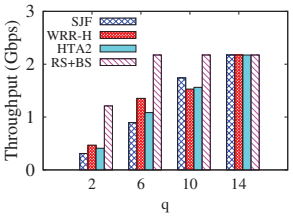


Fig. 13: Throughput vs. q with Robustness Constraints

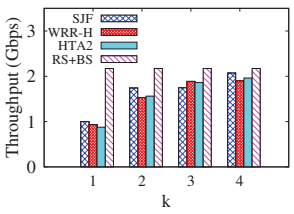


Fig. 14: Throughput vs. k with Robustness Constraints

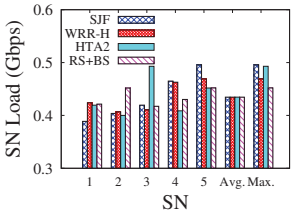


Fig. 11: Load of each SN without Robustness Constraints

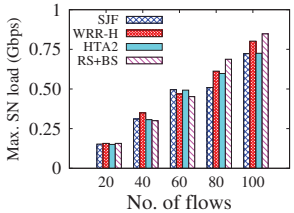


Fig. 12: Maximum SN Load of SNs vs. No. of Flows without Robustness Constraints

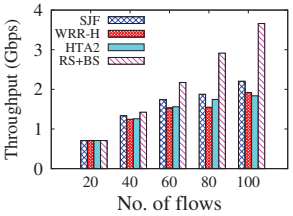


Fig. 15: Throughput vs. No. of Flows with Robustness Constraints

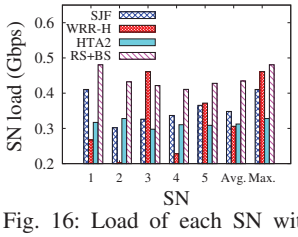


Fig. 16: Load of each SN with Robustness Constraints

over, we use vnStat tool [52] to monitor and collect the traffic information in service nodes.

*2) **Performance Comparison without Robustness Constraints**:* The results are shown in Figs. 9-12. We first generate 60 flows in total scattered among 20 tenants and send these flows to service nodes based on the results of different algorithms. We record the number of tenants that each service node serves and the results are shown in Fig. 9. We can see that RS+BS reduces the average number of tenants that a service node serves by 34%, 45% and 31.3% compared with SJF, WRR-H and HTA2, respectively. Fig. 10 shows that MNAS grows when the number of flows increases for all benchmarks. In contrast, our proposed algorithm only affects a limited range of service nodes. For example, RS+BS reduces MNAS by about 50%, 60% and 50% compared with SJF, WRR-H and HTA2, respectively, given 80 flows in the system. Although RS+BS takes the robustness constraints into consideration, Figs. 10-12 show that RS+BS can acquire similar service node load compared with the other three benchmarks.

From these results, we can see that our proposed algorithm can reduce the number of affected tenants when encountering service node failure and can reduce the number of affected service nodes when a tenant sends malicious traffic to mapped service nodes. Meanwhile, RS+BS can achieve similar load performance compared with other algorithms while achieving better network robustness.

*3) **Performance Comparison with Robustness Constraints**:* The results are shown in Figs. 13-16. The service node will refuse to serve those flows that violate the robustness constraints. As shown in Figs. 13-14, the throughput increases with the increasing k and q for all algorithms. When k and q are relatively small, the number of abandoned flows is significant for benchmarks and RS+BS can serve more traffic than other algorithms. For example, when $q = 6$ in Fig. 13, our proposed algorithm can improve the throughput by about 124.4%, 60.7% and 100.2% compared with SJF, WRR-H and HTA2, respectively. Fig. 15 shows the throughput increases with the number of flows for all algorithms, and RS+BS can achieve the maximum total load compared with other algorithms. For example, when the number of flows is 80, RS+BS improves the throughput by about 55.2%, 88.1% and 85.6% compared with SJF, WRR-H and HTA2, respectively. Fig. 13 shows the load of each service node when the number of flows is 60. It is obvious that RS+BS has the best average and maximum service node load performance.

From the above system evaluation results, we can draw some conclusions. First, Figs. 9-10 show that RS+BS can significantly reduce the number of affected tenants and the number of affected service nodes by about 40% and 55% on average, respectively, compared with the other three benchmarks. Second, RS+BS can obtain a similar service node load performance compared with the other three algorithms while our proposed algorithm considers robustness constraints as shown in Figs. 11-12. Finally, Figs. 13-16 indicate that RS+BS can always acquire the highest load among all service nodes if all algorithms consider the robustness constraints.

VI. Conclusion

In this paper, we focus on the problem of robust service mapping in multi-tenant clouds. To efficiently solve this complex problem, we take a two-step approach: service node assignment and tenant traffic scheduling. Several algorithms with bounded approximation factors have been designed to solve the service node assignment problem and the tenant traffic scheduling problem. Extensive simulation results show the high efficiency of our proposed algorithms.

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; and in part by a research funding from Huawei.

REFERENCES

[1] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle, "Multi-tenant soa middleware for cloud computing," pp. 458–465, 2010.

[2] G. Peng, H. Wang, J. Dong, and H. Zhang, "Knowledge-based resource allocation for collaborative simulation development in a multi-tenant cloud computing environment," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 306–317, 2016.

[3] "The amazon web service," https://aws.amazon.com/.

[4] "The google cloud platform," https://cloud.google.com/.

[5] H. Deng, L. Huang, H. Xu, X. Liu, P. Wang, and X. Fang, "Revenue maximization for dynamic expansion of geo-distributed cloud data centers," *IEEE Transactions on Cloud Computing*, 2018.

[6] D. Shue, M. J. Freedman, and A. Shaikh, "Performance isolation and fairness for multi-tenant cloud storage," pp. 349–362, 2012.

[7] H. AlJahdali, A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu, "Multi-tenancy in cloud computing," in *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, pp. 344–351.

[8] M. Rahman, S. Iqbal, and J. Gao, "Load balancer as a service in cloud computing," in *2014 IEEE 8th International Symposium on Service Oriented System Engineering*. IEEE, 2014, pp. 204–211.

[9] 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 Computer Communication Review*, vol. 48, no. 1, pp. 49–55, 2018.

[10] M. Liu, W. Dou, S. Yu, and Z. Zhang, "A clusterized firewall framework for cloud computing," in *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014, pp. 3788–3793.

[11] K. Yang and X. Jia, "Data storage auditing service in cloud computing: challenges, methods and opportunities," *World Wide Web*, vol. 15, no. 4, pp. 409–428, 2012.

[12] J. Li, W. Shi, P. Yang, and X. Shen, "On dynamic mapping and scheduling of service function chains in sdn/nfv-enabled networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[13] M. Jalalitabar, G. Luo, C. Kong, and X. Cao, "Service function graph design and mapping for nfv with priority dependence," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–5.

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

[15] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, "Joint optimization of service function chaining and resource allocation in network function virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, 2016.

[16] S. Abrishami, M. Naghibzadeh, and D. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[17] A. Verma and S. Kaushal, "Deadline constraint heuristic-based genetic algorithm for workflow scheduling in cloud," *International Journal of Grid and Utility Computing*, vol. 5, no. 2, pp. 96–106, 2014.

[18] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," vol. 41, no. 4, pp. 350–361, 2011.

[19] A. Engelmann and A. Jukan, "A reliability study of parallelized vnf chaining," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[20] C. Delimitrou and C. Kozyrakis, "Bolt: I know what you did last summer... in the cloud," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 599–613, 2017.

[21] B. Grobauer, T. Walloschek, and E. Stocker, "Understanding cloud computing vulnerabilities," *IEEE Security & privacy*, vol. 9, no. 2, pp. 50–57, 2010.

[22] A. Ledjiar, E. Sampin, C. Talhi, and M. Cheriet, "Network function virtualization as a service for multi-tenant software defined networks," in *2017 Fourth International Conference on Software Defined Systems (SDS)*. IEEE, 2017, pp. 168–173.

[23] A. Shieh, S. Kandula, A. G. Greenberg, and C. Kim, "Seawall: Performance isolation for cloud datacenter networks."

[24] A. Leivadeas, M. Falkner, I. Lambadaris, and G. Kesidis, "Dynamic traffic steering of multi-tenant virtualized network functions in sdn enabled data centers," in *2016 IEEE 21st International Workshop on Computer Aided Modelling and Design of Communication Links and Networks*, pp. 65–70.

[25] D. Shan, F. Ren, P. Cheng, R. Shu, and C. Guo, "Observing and mitigating micro-burst traffic in data center networks," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 98–111, 2019.

[26] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[27] L. Mai, C. Hong, and P. Costa, "Optimizing network performance in distributed machine learning," in *7th {USENIX} HotCloud*, 2015.

[28] G. Sun, Z. Chen, H. Yu, X. Du, and M. Guizani, "Online parallelized service function chain orchestration in data center network," *IEEE Access*, vol. PP, no. 99, pp. 1–1, 2019.

[29] "Google cluster-data," https://github.com/google/cluster-data.

[30] Y. Ren, G. Liu, V. Nitu, W. Shao, R. Kennedy, G. Parmer, T. Wood, and A. Tchana, "Fine-grained isolation for scalable, dynamic, multi-tenant edge clouds," in *2020 {USENIX}{ATC}*, 2020, pp. 927–942.

[31] Y. Zhou, L. Ruan, L. Xiao, and R. Liu, "A method for load balancing based on software defined network," *Advanced Science and Technology Letters*, vol. 45, pp. 43–48, 2014.

[32] Z. Liu, K. Chen, H. Wu, S. Hu, Y.-C. Hut, Y. Wang, and G. Zhang, "Enabling work-conserving bandwidth guarantees for multi-tenant datacenters via dynamic tenant-queue binding," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 1–9.

[33] W. J. A. Silva, "Avoiding inconsistency in openflow stateful applications caused by multiple flow requests," in *2018 ICNC*, pp. 548–553.

[34] A. Gemberjacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: enabling innovation in network function control," vol. 44, no. 4, pp. 163–174, 2015.

[35] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, and H. Wang, "Incremental deployment and throughput maximization routing for a hybrid sdn," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1861–1875, 2017.

[36] H. Moens and F. De Turck, "Vnf-p: A model for efficient placement of virtualized network functions," in *10th International Conference on Network and Service Management (CNSM) and Workshop*. IEEE, 2014, pp. 418–423.

[37] E. Sanlaville and G. Schmidt, "Machine scheduling with availability constraints," *Acta Informatica*, vol. 35, no. 9, pp. 795–811, 1998.

[38] L.-C. Chen and H.-A. Choi, "Approximation algorithms for data distribution with load balancing of web servers." in *cluster*, vol. 1, 2001, p. 274.

[39] H. Xu, H. Huang, S. Chen, G. Zhao, and L. Huang, "Achieving high scalability through hybrid switching in software-defined networking," *IEEE/ACM Transactions on Networking*, vol. PP, no. 1, pp. 1–15, 2018.

[40] IBM, "V12. 1: User manual for cplex," *International Business Machines Corporation*, vol. 46, 01 2009.

[41] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on sdn network utilization," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 1734–1742.

[42] H. Xu, Z. Yu, X.-Y. Li, C. Qian, L. Huang, and T. Jung, "Real-time update with joint optimization of route selection and update scheduling for sdns," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–10.

[43] J. K. Lenstra, D. B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," in *IEEE FOCS*, Oct 1987, pp. 217–224.

[44] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 2211–2219.

[45] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[46] Karakostas and George, "Faster approximation schemes for fractional multicommodity flow problems," *Acm Transactions on Algorithms*, vol. 4, no. 1, pp. 1–17, 2008.

[47] J. Nievergelt and E. M. Reingold, "Binary search trees of bounded balance," *SIAM journal on Computing*, vol. 2, no. 1, pp. 33–43, 1973.

[48] R. Somula and S. Nalluri, "Analysis of cpu scheduling algorithms for cloud computing," 2019.

[49] N. K. C. Das, M. S. George, and P. Jaya, "Incorporating weighted round robin in honeybee algorithm for enhanced load balancing in cloud environment," 2017.

[50] Keke, Gai, Meikang, Qiu, Hui, and Zhao, "Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing," *Journal of Parallel and Distributed Computing*, 2018.

[51] "iperf," https://iperf.fr/.

[52] "vnstat," https://humdi.net/vnstat/.