VITA: Virtual Network Topology-aware Southbound Message Delivery in Clouds

Luyao Luo1,2 Gongming Zhao1,2 Hongli Xu1,2 Liguang Xie3 Ying Xiong3
1School of Computer Science and Technology, University of Science and Technology of China
2Suzhou Institute for Advanced Research, University of Science and Technology of China
3Futurewei Technologies, USA

Abstract—Southbound message delivery from the control plane to the data plane is one of the essential issues in multi-tenant clouds. A natural method of southbound message delivery is that the control plane directly communicates with compute nodes in the data plane. However, due to the large number of compute nodes, this method may result in massive control overhead. The Message Queue (MQ) model can solve this challenge by aggregating and distributing messages to queues. Existing MQ-based solutions often perform message aggregation based on the physical network topology, which do not align with the fundamental requirements of southbound message delivery, leading to high message redundancy on compute nodes. To address this issue, we design and implement VITA, the first-of-its-kind work on virtual network topology-aware southbound message delivery. However, it is intractable to optimally deliver southbound messages according to the virtual attributes of messages. Thus, we design two algorithms, submodular-based approximation algorithm and simulated annealing-based algorithm, to solve different scenarios of the problem. Both experiment and simulation results show that VITA can reduce the total traffic amount of redundant messages by 45%-75% and reduce the control overhead by 33%-80% compared with state-of-the-art solutions.

Index Terms—Southbound Message Delivery, Message Queue, Virtual Network Topology, Virtual Private Cloud.

I. INTRODUCTION

Nowadays, as more enterprise customers migrate their on-premise workloads to the cloud, the user base of a cloud provider overgrows in just a few years [1]. In current cloud deployment model, tenants deploy virtual machines (VMs) on compute nodes in the cloud data plane and manage the VMs through unified restful APIs by the cloud control plane [2]. The control plane processes tenants’ requests, and sends network configuration messages, also called southbound messages, to computes nodes [3]. Over the past decade, we are observing rapid growth of the number of customers and the continuous expansion of individual network size. As a result, the number of southbound messages is mounting a rapid pace [4]. Thus, how to deliver the southbound messages with low provisioning latency and low control overhead has become a critical issue for hyper-scale cloud deployments [5]–[7].

A natural method to deliver southbound messages is direct end-to-end transmission via message passing interfaces (MPI) [8] or remote procedure call (RPC) [9]. For example, as one of the common protocols in distributed microservice frameworks, RPC establishes TCP links between servers and clients. In this way, each compute node directly communicates with controllers and receives all the required messages.

The downside is that, as the network scale increases, the direct communication method will cause a high load on the control plane, leading to message congestion or loss, especially when encountering burst southbound traffic [10]. This insight has been discovered by the experiments [11], in which gRPC [12] and Apache Thrift [13], two widely used open-source RPC frameworks, are tested. The results show that when the payload size of each message increases from 1KB to 10KB without limitation on the sending rate, the successful queries per second drops from 10K to 4K.

Therefore, it is necessary to reduce southbound control overhead in a large-scale cloud by decoupling the data plane from the control plane [14, 15]. As an alternative, the Message Queue (MQ) model is one of the most widely adopted messaging solutions used to build cloud infrastructure and tenant applications in the cloud [16, 17]. Specifically, a MQ server is used as a messaging middlebox between the control plane and the data plane, which implements multiple queues for storing and forwarding messages. Each queue is responsible for forwarding a set of messages with the same attributes (e.g., subnet). Under this model, the controller sends messages to different queues according to message attributes, while compute nodes receive messages in one or more queues by their own needs [18, 19]. The key step in the MQ model is to determine which queues the controller should send each message to, and which queues each compute node receives messages from.

One of the most intuitive ideas inside the MQ model is to specify a queue for each compute node. That is, the messages are classified at the granularity of a single computing node. In this way, the control plane sends each message to an exclusive queue, and the corresponding computing node can obtain the message by subscribing to the corresponding queue. However, in reality, compared to a large number of compute nodes (such as 5,500 compute nodes in CERN [4]), a message queue server commonly supports a relatively small number of queues. For example, the experiments of Apache Kafka (a well-known open-source message queue) from [20] show that setting up a few hundred queues will lead to frequent crashes of the message queue server. Therefore, messaging at the granularity of a single compute node is not feasible in a large-scale cloud, and we must carry out message aggregation with a proper granularity.

A common way for message aggregation is Node Grouping (NG) in OpenStack Nova [21]. That is, the compute nodes are
In this paper, we design a virtual network topology-aware southbound message delivery system, called VITA. Specifically, we use VPC as the granularity to aggregate southbound messages. At the same time, considering a large number of VPCs, how to aggregate messages of these VPCs into a limited number of message queues with both low control overhead and low message redundancy is also very difficult. To solve this issue, we propose two algorithms, submodular based approximation algorithm and simulated annealing based algorithm, to solve different scenarios. Both experiment and simulation results show that VITA dramatically reduces the total traffic amount of redundant messages by 45%-75% and reduces the control overhead by 33%-80% compared with state-of-the-art solutions.

II. MOTIVATION AND VITA OVERVIEW

A. A Motivation Example

This section gives an example to illustrate the pros and cons of both RPC and NG. A simple example of southbound message delivery is illustrated in Fig. 1. There are 1 controller, 4 compute nodes and 3 VPCs in the cloud. The VMs of 3 VPCs are deployed on compute nodes. Specifically, VMs of VPC 1 are deployed on compute nodes \( n_1 \) (VM1-1), \( n_2 \) (VM1-2, VM1-3) and \( n_3 \) (VM1-4). VMs of VPC 2 are deployed on compute nodes \( n_1 \) (VM2-1), \( n_2 \) (VM2-2, VM2-3) and \( n_4 \) (VM2-4). VMs of VPC 3 are deployed on nodes \( n_2 \) (VM3-2), \( n_3 \) (VM3-3) and \( n_4 \) (VM3-4). For ease of explanation, we assume that the control plane will send a network configuration message for each VPC. The performance results are summarized in Table I.

RPC establishes connections between the controller and all compute nodes in Fig. 1(a). If a message will be sent to a VPC, the controller sends this message to the destination nodes, which contain VMs of this VPC, in turn. A mapping table is maintained in the database to record the mapping relationship between the VPCs and the compute nodes. To realize the southbound message delivery, the controller queries this table and determines compute nodes to which the messages should be sent. For example, to process the
configuration message of VPC 1, the controller queries the database and obtains the IP addresses of compute nodes (i.e., n1, n2 and n3). Then, the controller will send the configuration messages to these three nodes through RPC. As a result, the controller sends 10 messages in total and the data plane receives 10 messages accordingly.

The Node Grouping (NG) method divides the four compute nodes into two groups, as shown in Fig. 1(b), and uses a message queue server for storing and forwarding messages. All the queues are identified by topics. The controller sends message to one queue by publishing messages to a topic, and each compute node receives messages from one queue by subscribing to a topic. The MQ server in this example contains two queues, which are identified by topics group1 and group2, respectively. On processing the configuration message of VPC 1, the controller queries the database and obtains the nodes which require this message. The nodes n1 and n3 are in group 1 and group 2, separately. So, the controller should send two messages with the same content to the MQ server. One is published to topic group1, and the other is to topic group2. In all, the controller sends 6 messages in total. However, as the compute nodes in the same group will receive all the messages from a queue, a node will receive some invalid messages. For example, node n2 receives 3 messages of VPCs 1, 2, and 3, but only 2 messages from VPCs 1 and 3 are necessary. Node n4 receives 3 messages with 1 unnecessary message of VPC 1. As a result, all the compute nodes in the data plane receive 12 messages, 2 of which are unnecessary.

B. Our Intuition

We observe that the two solutions of southbound message delivery have advantages and disadvantages. RPC allows each compute node to receive only the required messages without any redundancy. In small-scale clouds, perhaps this is the most proper solution. However, in large-scale distributed cloud scenarios, the pressure of the control plane will be weighty, and the message delivery latency may be very high [11]. As for the node grouping solution, the pressure of the control plane can be reduced while the load on the data plane (redundant messages) significantly increases.

A question immediately following the above discussion is that can we do better by using MQ with less redundant messages and low control overhead? Clearly, we should use as many queues as possible for southbound message delivery. However, too many queues will lead to frequent crashes of the message queue server [20]. Therefore, how to effectively aggregate many messages into a limited number of queues is necessary. As mentioned above, southbound messages have not only physical attributes (e.g., IP address of the destination node) but also virtual attributes (e.g., VPC ID) under the virtual private cloud architecture. Moreover, messages from the same VPC are more likely to be sent to the same virtual address in the virtual network [26, 27]. In other words, aggregating and delivering southbound messages according to the attributes of the VPC is more intuitive and efficient than existing solutions.

As shown in Fig. 1(c), since there are 3 VPCs and 2 queues in this example, the controller aggregates the messages of VPCs 1 and 2, and sends these messages to the same queue (with topic vn1). Meanwhile, the controller sends the messages of VPC 3 to another queue (with topic vn2). Each compute node subscribes to different topic(s) according to the messages it needs. For example, because node n2 only needs the messages of VPCs 1 and 2, it only subscribes to topic vn1. Similarly, since node n4 needs the messages of VPCs 2 and 3, it should subscribe to both topics vn1 and vn2. Accordingly, the controller sends 3 messages, and all the compute nodes in the data plane totally receive 11 messages, 1 out of which is unnecessary. As a result (shown in Table 1), this scheme achieves lower control overhead compared with RPC, and achieves better data/control plane performance compared with NG. Motivated by this example, we design a virtual network topology-aware southbound message delivery scheme, called VITA.

<table>
<thead>
<tr>
<th>schemes</th>
<th>n1</th>
<th>n2</th>
<th>n3</th>
<th>n4</th>
<th>data plane</th>
<th>control plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>NG</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>VITA</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>3</td>
</tr>
</tbody>
</table>

TABLE I: The number of messages received by each compute node, received by the data plane, and sent by the control plane through three delivery schemes.

Fig. 2: Overall system overview and workflow of VITA.

C. System Overview and Workflow of VITA

As shown in Fig. 2, VITA mainly consists of three parts: the control plane (composed of the controllers), the data plane (composed of the compute nodes), and the message queue server. Specifically, the control plane consists of a set of distributed microservices, and one of its functions is to manage the virtual network through southbound message delivery. In order to build the correspondence between VPCs and topics, a mapping table from VPCs to topics, instead of VPCs to IPs, is maintained. We will describe in detail how to determine the correspondence in Section III. In the data plane, VMs belonging to different VPCs are distributed on different compute nodes. For more efficient implementation, a control agent is designed on each node to be responsible for subscribing to topics, distinguishing messages, and parsing requests. The agent manages all virtual machines on the node
and knows to which VPCs they belong. As an important component, the MQ server is responsible for the asynchronous communication between the control and data planes.

Fig. 2 also briefly describes the system workflow. The system process is mainly triggered by two events. One is the launch of a new VM on the compute node. When a VM is added or migrated, the control agent queries the database to get the required messages of different VPCs. The other one is the configurations by a tenant. When one tenant configures their VPC through provided API (e.g., subnet, security group), the control plane parses the request and constructs corresponding southbound messages. Then it queries the database and determines which topic(s) the messages should be published to. Next, the agent sends the messages to specified queues and asynchronously waits for the reply of the processing result. Finally, the agent receives messages from specific queues and judges whether it is valid or not according to the VPC ID of the message. If no VM needs this message, it will be discarded. Otherwise, the control agent will perform corresponding operations (e.g., setting IP, configuring routing table) on VMs according to the content of the message and return the operation result to the control plane. In this way, VITA can realize the decoupling of the control plane and the data plane.

III. PROBLEM FORMULATION AND ALGORITHM DESIGN

A. Network Models

A typical cloud consists of the control plane and the data plane. Specifically, a cluster of controllers constitute the control plane, and are responsible for managing the network, including southbound message delivery. The data plane consists of a set of compute nodes, and is responsible for providing computing resources for tenants. We use $N = \{n_1, n_2, ..., n_{|N|}\}$ to represent the set of compute nodes. The set of VPCs in the cloud is denoted as $V = \{v_1, v_2, ..., v_{|V|}\}$. Tenants create VPCs in the cloud by deploying VMs on compute nodes.

We adopt the MQ model to implement southbound message delivery. Specifically, an MQ server containing a set of queues, serves as the messaging middlebox in a cloud and adopts the publish/subscribe model [28, 29]. The queues are responsible for storing and forwarding southbound messages from the control plane to the data plane. Each queue is identified by a topic. When the controller sends messages to one queue, we say that the controller publishes messages to the topic. The compute nodes receive messages from a queue by subscribing to the corresponding topic. The topic set is defined as $T = \{t_1, t_2, ..., t_K\}$, where $K = |T|$ is the number of queues in the MQ server.

B. Problem Formulation

The section gives the formulation of the virtual network southbound message delivery (VSMD) problem. Specifically, we use VPC as the granularity to aggregate southbound messages. Due to the prior work of traffic matrix prediction in clouds [30, 31], it is reasonable to assume that we can obtain the expected traffic intensity of southbound messages for each VPC $v \in V$, which is denoted as $f(v)$.

The key step of VSMD is to determine to which queue(s) the controllers should deliver each message, and from which queues each compute node receives messages. Thus, we use binary variable $y^v_t$ to denote whether the controller will publish the messages of VPC $v$ to topic $t$ or not. Meanwhile, we use binary variable $z^v_n$ to represent whether the compute node $n$ will subscribe to topic $t$ or not.

In order to deliver southbound messages successfully, we should consider the following two constraints. 1) Each compute node must obtain all the required messages. That is, each compute node should receive the messages of VPC $v$ if a VM belonging to $v$ is deployed on this node. The constant $\Gamma^v_n$ indicates whether the compute node $n$ contains the VMs belonging to VPC $v$ or not. 2) The traffic amount of messages on each node should not exceed its capacity. We use $s(n)$ to denote the message processing capability of node $n$. Once a compute node subscribes to a topic, it will receive all the messages in this queue, which results in message redundancy. Thus, our objective is to minimize the total traffic amount on compute nodes (or in the data plane).

We give the following problem definition:

$$\min \sum_{n \in N} b(n)$$

$$\begin{align*}
\sum_{t \in T} y^v_t & \geq 1, & \forall v \in V \\
\sum_{t \in T} z^v_n y^v_t & \geq \Gamma^v_n, & \forall n \in N, v \in V \\
\sum_{v \in V} \sum_{t \in T} z^v_n y^v_t f(v) & = b(n), & \forall n \in N, v \in V \\
y^v_t, z^v_n & \in \{0, 1\}, & \forall v, n, t
\end{align*}$$

The first set of inequalities indicates that each VPC subscribes to at least one topic. The second set of inequalities represents that all the VMs on any compute node should receive all the required messages. The third set of equalities shows the message traffic amount on each compute node $n$, denoted as $b(n)$. The fourth set of inequalities expresses the message processing capacity constraint on each compute node $n$. Our objective is to minimize the total message traffic amount on compute nodes, that is, $\min \sum_{n \in N} b(n)$.

**Theorem 1** The VSMD problem is NP-hard.

We prove the NP-hardness by showing that the Weighted Set Covering Problem (WSCP) [32] is a special case of VSMD. Due to space limit, we omit the detailed proof here.

C. Algorithm Design for VSMD

1) Algorithm Overview: If the controller sends the messages of each VPC to only one queue, the total traffic amount of messages delivered by the controller can be minimized. Considering that the controller is often the bottleneck in a cloud, it is reasonable to assume that messages of each VPC are sent to only one queue. To deal with this scenario, this section presents a submodular-based approximation algorithm to solve the VSMD problem. We will consider the scenario where the messages of each VPC can be forwarded to more than one queue in the next section.
In this section, we regard that the messages of each VPC are sent to only one queue. As a result, the VPC set can be divided into $K$ subsets, and each VPC in the same subset is assigned with the same topic. Initially, all VPCs belong to the same set. Our algorithm consists of $K$ iterations where $K$ is the number of queues (i.e., the number of topics) in the MQ server. In each iteration, we determine a subset of $V$ that can reduce the total traffic amount of messages the most and assign all the VPCs in this subset with one topic.

2) Preliminaries: We first give the definition of the traffic amount of messages of VPC set $V'$ as follows:

**Definition 1** For any VPC set $V'$, the total traffic amount of messages of all the VPCs in $V'$ is

$$R(V') = |Sub(V')| \sum_{v \in V'} f(v)$$

(2)

where $Sub(V')$ is the set of compute nodes which contain VMs belonging to any VPC $v \in V'$.

We need to divide the VPCs into $K$ sets so that messages of each VPC will be published to one of $K$ topics. Initially, when all the VPCs belong to one set, the total traffic amount of messages on all compute nodes can be expressed as $R(V) = |N| \cdot \sum_{v \in V} f(v)$, where $|N|$ is the number of compute nodes. If we divide VPCs into $K$ sets, denoted as $\{V_1, V_2, ..., V_K\}$, the traffic amount of all southbound messages becomes $\sum_{i=1}^{K} R(V_i)$. In other words, the traffic amount of messages will be reduced as much as possible by dividing VPCs into $K$ sets. That means the minimization problem in Eq. (1) can be converted into the following equivalent maximization problem:

$$\max \ R(V) = \sum_{i=1}^{K} R(V_i)$$

(3)

Subject to:

$$\begin{cases}
\sum_{i \in I} y_i \geq 1, & \forall v \in V \\
\sum_{i \in I} \zeta_{i} y_i \geq \Gamma_{v}, & \forall n \in N, v \in V \\
\sum_{i \in I} \sum_{v \in V} \gamma_{i} y_i f(v) \leq s(n), & \forall n, v, t
\end{cases}$$

This problem is similar to a clustering problem, where we need to divide the VPC set $V$ into $K$ clusters to maximize the traffic amount reduction on compute nodes. Our algorithm is based on efficient computations of a submodular set function $\varphi$, which defines the maximum traffic amount reduction of messages by dividing the VPCs into several sets. We give the definition of the submodular set function $\varphi$ as follows.

**Definition 2** Given the set $\Phi$, which contains disjoint subsets of $V$, the traffic amount reduction of messages achieved by dividing the VPCs according to $\Phi$ is defined as:

$$\varphi(\Phi) = R(V) - \sum_{S \in \Phi} R(S) - R(V - M)$$

(4)

where $M$ is the set of VPCs that can be covered by all the sets in $\Phi$. That is, $M = \bigcup_{S \in \Phi} S$.

Next, we give the definition of submodularity, and prove that the function $\varphi$ is submodular.

**Definition 3** (Submodularity): Given a finite set $E$, a real-valued function $z$ on the set of subsets of $E$ is called submodular if $z(S \cup \{e\}) - z(S) \leq z(S' \cup \{e\}) - z(S')$ for all $S' \subseteq S \subseteq E$ and $e \in E - S$.

**Lemma 2** Given the set $U$ as the power set of $V$, the function $\varphi$ defined in Eq. (4) is submodular on $U$.

Proof: Without loss of generality, we consider an arbitrary set $\Phi \subseteq U$ and an arbitrary set $A \subseteq V$. Assume that $A$ does not intersect with other sets in $\Phi$, i.e., $A \cap S = \emptyset, \forall S \in \Phi$. Then, we have

$$\varphi(\Phi \cup \{A\}) - \varphi(\Phi) = R(V - M) - R(V - M - A) - R(A)$$

(5)

where $M = \bigcup_{S \in \Phi} S$. Given an arbitrary subset $\Phi' \subseteq \Phi$, it also follows

$$\varphi(\Phi' \cup \{A\}) - \varphi(\Phi') = R(V - M') - R(V - M' - A) - R(A)$$

(6)

where $M' = \bigcup_{S \in \Phi'} S$.

Note that $R(V - M) - R(V - M - A) - R(A)$ represents the traffic amount reduction by dividing set $V - M$ into two subsets: $V - M - A$ and $A$. Since $\Phi'$ is the subset of $\Phi$, $V - M$ is the subset of $V - M'$ accordingly. Thus, we have:

$$R(V - M) - R(V - M - A) \leq R(V - M') - R(V - M' - A)$$

(7)

Combining Eqs. (5), (6) and (7), we know that:

$$\varphi(\Phi \cup \{A\}) - \varphi(\Phi) \leq \varphi(\Phi' \cup \{A\}) - \varphi(\Phi')$$

(8)

According to Definition 3, we show that the set function $\varphi$ is submodular.

To maintain the processing capacity constraint of a single compute node $n$, i.e., $b(n) \leq s(n)$, we only focus on the set $A \subseteq V$ without breaking the constraint, that is,

$$\sum_{v \in A} f(v) \leq \min_{n \in Sub(A)} s(n)$$

(9)

We call the sets satisfying Eq. (9) as feasible sets. The feasible sets can be explored efficiently by simply performing a depth-first search [33] on the VPC set $V$. We omit the detailed description here due to space limit.

3) Algorithm Description: Given these insights, we propose the submodular-based southbound message delivery algorithm (SM-SMD) in detail, which is formally described in Alg. 1. SM-SMD consists of three steps. In the first step, the algorithm computes a set of feasible sets $S$ in advance and starts with an empty set $\Phi$ (Line 3). In the second step (Lines 5-12), it loops through the possible feasible set $S \in \Phi$ to find the maximum function value $\varphi(\Phi \cup \{S\})$. The algorithm performs $K - 1$ iterations until we obtain $K$ sets of VPCs. In the third step (Lines 13-17), we obtain the mapping relationship between VPCs and topics (i.e., $y_i$).

4) Performance Analysis: We analyze the approximation performance of our proposed algorithm based on the following lemma.

**Lemma 3** For a real-valued submodular and non-decreasing function $z(S)$ on $U$, the optimization problem $\max_{S \subseteq U} z(S) : |S| \leq K, z(S) \text{ is submodular}$ can reach a $(1-1/e)$ approximation factor if the algorithm performs greedily [34].

**Theorem 4** Our SM-SMD achieves a $(1-1/e)$ approximation factor for the maximization problem in Eq. (3).

Proof: The function $\varphi$ is submodular by Lemma 2. Besides, for any set $\Phi$ of subsets of $V$ and $A \subseteq V$ with
Algorithm 1 SM-SMD: Submodular-based Algorithm for VSMD

1: **Step 1: Initialization**
2: Compute the set of feasible sets \( \mathcal{S} \)
3: \( \Phi \leftarrow \emptyset \)
4: **Step 2: Greedy Selection**
5: while \(|\Phi| \leq K - 1\) do
6: \( \text{Set } tmp \leftarrow 0, \text{opt} \leftarrow 0 \)
7: for \( S \in \mathcal{S} - \Phi \) do
8: \( \text{tmp } \leftarrow \varphi(\Phi \cup \{S\}) \)
9: if \( \text{tmp } > \text{opt} \) then
10: \( \text{opt } \leftarrow \text{tmp}, S^* \leftarrow S \)
11: \( \Phi \leftarrow \Phi + \{S^*\} \)
12: \( \Phi \leftarrow \Phi + \{V - \bigcup_{S \in \Phi} S\} \)
13: **Step 3: Assignment of VPCs and Topics**
14: \( i \leftarrow 1 \)
15: for \( S \in \Phi \) do
16: \( \text{Set } y_v^i = 1 \text{ if } v \in S \)
17: \( i \leftarrow i + 1 \)

We determine the parameters based on the work [38] to achieve a high probability for converging to the global optimal solution.

SA-SMD first initializes the parameters and the initial state. As SM-SMD can obtain a feasible assignment of VPCs and topics, SA-SMD takes the results of SM-SMD as the initial state. Then it executes a two-level iteration. In the each round of the inner iteration (Lines 4-11), the algorithm randomly selects a VPC and a topic to change their mapping relationship (i.e., \( y_v^i = 1 - y_v^i \)) (Lines 6-7) and calculates the difference in the total traffic amount of messages on all compute nodes by re-selecting topics, denoted as \( \Delta \) (Line 8). If \( \Delta \leq 0 \), it means that the message redundancy is reduced, and we accept the current state. Otherwise, we refuse the current state with probability \( 1 - e^{-\frac{\Delta}{\alpha}} \) (Lines 9-10).

Each inner iteration runs in \( L \) rounds. In the outer iteration, temperature \( T \) is decreased by a factor \( \alpha \) at the end of the inner iteration (Line 13). Then, if \( T \geq t_m \), the algorithm terminates and outputs the final result. Otherwise, it performs a new inner iteration with a decreased temperature. The SA-SMD algorithm is formally described in Alg. 2.

Algorithm 2 SA-SMD: Simulated Annealing based Algorithm for VSMD

1: Input \( L, t_0, t_m, \alpha \)
2: Run SM-SMD to obtain the solution: \( y_v^i \) and \( z_v^i = \Gamma_v y_v^i \)
3: Init temperature \( T = t_0, k = 0 \)
4: while \( T \geq t_m \) do
5: while \( k \leq L \) do
6: Select a random VPC \( v \) and a random topic \( t \)
7: \( y_v^i \leftarrow 1 - y_v^i \)
8: \( \text{Set } \Delta \text{ to be difference of total traffic amount by topic re-selection.} \)
9: if \( \Delta > 0 \) then
10: \( \text{Set } y_v^i \leftarrow 1 - y_v^i \text{ with probability } 1 - e^{-\frac{\Delta}{\alpha}} \)
11: \( k \leftarrow k + 1 \)
12: \( \text{Set } k = 0 \)
13: \( T \leftarrow \alpha T \)
14: Output the results

In each round of the inner iteration, the algorithm calculates the difference of traffic amount received by each compute node by re-selecting topics, which costs \( O(|N|) \) time. This calculation loops \( L \) times at each temperature \( T \), which drops from \( t_0 \) to \( t_m \) at the decreasing rate of \( \alpha \). Thus, we execute the calculation for \( \log_\alpha(t_m/t_0) \) times and the overall time complexity of SA-SMD is \( O(L \cdot \log_\alpha(t_m/t_0) \cdot |N|) \).

IV. Performance Evaluation

A. Performance Metrics and Benchmarks

This paper studies how to deliver southbound messages in clouds with low control overhead and low message redundancy. The code is open-source and available at https://github.com/futurewei-cloud/vita. We adopt five main metrics for performance evaluation. (1) The **control overhead** represents the resource consumption of the controller for southbound
message delivery. In the testbed experiment, we measure the controller's CPU utilization during system running as the control overhead. Meanwhile, we record the total traffic amount of messages sent by the controller as the control overhead in large-scale simulations. (2) The MQ overhead indicates the resource consumption of the MQ server to process southbound messages. According to [20], disk I/O utilization is the main performance bottleneck of the MQ server. Thus, we use disk I/O utilization as the MQ overhead in the testbed experiment. As for large-scale simulations, we measure the total traffic amount of the messages through the MQ server as the MQ overhead. (3) The total traffic amount of all compute nodes. We measure the total traffic amount of southbound messages received by each compute node, and calculate the total (or maximum) value of all compute nodes as the third (or the fourth) metric. (5) The average message delivery delay. We record the time interval from the controller sending the southbound message to the compute node receiving the message as the message delivery delay. We compute the average delivery delay of all messages during the system running as this metric.

In this paper, we propose two message aggregation and distribution algorithms, SM-SMD and SA-SMD, based on VITA. We denote the corresponding schemes as VITA-SM and VITA-SA, respectively. To evaluate the performance of our VITA-SM and VITA-SA, we choose the following three state-of-the-art solutions as benchmarks.

1) The first one is RPC [9], which is a widely used method in distributed microservice framework for communications between servers and clients. In clouds, RPC establishes TCP connections between the controller and all compute nodes. Messages are sent from the controller to corresponding compute nodes one by one.

2) The second one is NG [21], which performs southbound message delivery using message queues. To deal with a limited number of message queues on the server, compute nodes are divided into certain groups according to a certain attribute (such as physical location). The nodes in the same group will subscribe to a same topic (i.e., queue) and receive the same messages.

3) The third one is denoted as VITA-KM. Since there is no exact work about southbound message delivery based on virtual network topology, we use the classic clustering algorithm, K-means [40], to aggregate and distribute messages with VPC as the granularity, VITA-KM takes the number of topics as the input \( k \), and divides the set of VPCs into \( k \) clusters.

We refer to a practical private cloud deployed in CERN (European Organization for Nuclear Research) [4] to design our simulation. The CERN private cloud contains 5,500 compute nodes. We change the scale of the virtual network by varying the number of VPCs from \( 1 \times 10^5 \) to \( 9 \times 10^5 \). We assume that the VMs are distributed on the compute nodes randomly, and the number of topics is set to 1,100 by default. As a result, NG divides the compute nodes into 1,100 groups, and each group contains 5 compute nodes. The expected message traffic intensity for each VPC is set as 1Mbps. Moreover, we use power law for the message-size distribution, where 20% of all messages account for 80% of traffic volume as observed in [41].

We observe the control overhead, the MQ overhead, and the total/maximum traffic amount on compute nodes by changing the number of VPCs in the cloud. The results are shown in Figs. 3-6. Specifically, Fig. 3 shows that the control overhead of all solutions increases with the increasing number of VPCs, and the growth rate of VITA-based solutions is significantly slower than that of RPC and NG. For example, given \( 7 \times 10^5 \) VPCs, the control overheads of VITA-SM, VITA-KM, and VITA-SA are 3.6Gbps, 3.6Gbps, and 6.8Gbps, respectively, while those of RPC and NG are 35.1Gbps and 17.5Gbps, respectively. It means that VITA-based solutions can reduce the control overhead by over 80% and 60% compared with RPC and NG, respectively. That is because the more messages delivered by the controller, the higher its control overhead. Specifically, the controller directly communicates with compute nodes by RPC, and each compute node only receives the required messages. NG reduces the control overhead by 50.1% compared with RPC by adopting the MQ model but still results in a higher control overhead compared with VITA-based solutions. The reason is the nodes are grouped based on the physical network topology, resulting in significant differences in required messages of nodes in the same group. As for three VITA-based solutions, both VITA-SM and VITA-KM can reduce the control overhead by about 47% compared with VITA-SA. That is because VITA-SA may send the same message to multiple queues, while VITA-SM and VITA-KM only send each message to exactly one queue.

Fig. 4 shows the MQ overhead of NG and three VITA-based algorithms by changing the number of VPCs. Note that we do not evaluate this metric for RPC since RPC does not use the MQ model. The results of the MQ overhead are of a similar trend with those of the control overhead for these algorithms. That is because both control overhead and
MQ overhead are positively correlated with the total traffic amount of southbound messages. For instance, when there are $5 \times 10^4$ VPCs, the MQ overheads of VITA-SM, VITA-KM, and VITA-SA are 2.5Gbps, 2.5Gbps, and 4.9Gbps, respectively, while that of RPC is 12.4Gbps. That is, both VITA-SM and VITA-KM can reduce the MQ overhead by about 79.8% and 60.5% compared with NG and VITA-SA, respectively.

Figs. 5-6 show that the total/maximum traffic amount on compute nodes increases for all solutions with the increasing number of VPCs. RPC and NG achieve the lowest and highest total/maximum traffic amount on compute nodes among all solutions, respectively. That is because RPC using the direct communication method will not cause message redundancy, while NG using a physical host-based grouping scheme will result in high redundancy. Note that, since RPC will cause an unacceptable control overhead as shown in Fig. 3, it is not feasible in large-scale clouds. We use the total/maximum traffic amount on compute nodes of RPC as the low bound to compare with other solutions. For example, given $6 \times 10^4$ VPCs in the cloud, the total traffic amount on compute nodes is 61Gbps, 65Gbps, and 90Gbps for VITA-SA, VITA-SM, and VITA-KM, respectively, while that of NG is 198Gbps. These results mean that VITA-SM reduces the total traffic amount on compute nodes by 29% and 66% compared with VITA-KM and NG, respectively, while slightly increases the traffic amount on compute nodes by 6% compared with VITA-SA. The total/max traffic amount on compute nodes of VITA-SA is lower than that of VITA-SM because it sends messages to more queues with higher control overhead to achieve lower message redundancy.

From these simulation results, we can draw some conclusions. First, as shown in Fig. 3, RPC is not feasible in large-scale clouds because it will cause unacceptable control overhead. Second, as shown in Figs. 3-6, VITA-based solutions can achieve superior performance, including lower control/MQ overhead and lower total/maximum traffic amount compared with NG. Third, VITA-SM reduces the total/maximum traffic amount by 29%/37% and achieves similar control/MQ overhead performance compared with VITA-KM. Fourth, compared with VITA-SA, VITA-SM reduces the control/MQ overhead by 47%/49% and increases the total/maximum traffic amount by 6%/15%.

C. System Implementation

1) Implementation on the Platform: In general, we use 10 servers running Ubuntu 18.04 with Linux kernel 5.4 to build the testbed. All the servers are equipped with a 22-core Intel Xeon 6152 processor, 128GB memory and an Intel X710 10GbE NIC. Among them, two servers are used as the controller and the message queue server, respectively. We take a small cloud deployed in GoDaddy [42] as a reference, which contains 350 compute nodes. We rely on the virtualization technology for system implementation to expand the testing topology and collect testing data conveniently. Specifically, we deploy 350 VMs, each equipped with 1 vCPU and 1GB memory, as compute nodes on the remaining 8 servers. The number of VPCs and topics is by default set to 300 and 100.

We run three sets of experiments on the platform. The expected traffic intensity for messages of each VPC is set to 1Mbps and the bandwidth constraint of each compute node is 1Gbps by default. The message-size distribution is the same as in simulations where 20% of all messages account for 80% of traffic amount. These messages are distributed in size from 512Bytes to 4MB. According to [43], we generate two types of messages: (1) unicast messages, whose sources and destinations are randomly picked, e.g., IP address segment configuration messages; (2) multicast messages, which simulate the traffic with multiple destinations, e.g., subnet and security group configuration messages. Each type of message accounts for half of the total traffic amount.

2) Test Results: The first set of experiments compares the overall performance of all benchmarks using three well-known MQ frameworks. Specifically, we take three open-source MQ frameworks for comparison: Apache Kafka (version 2.6.0) [44], RabbitMQ (version 3.8.19) [45], and Apache Pulsar (version 2.6.1) [46]. Kafka is the most widely deployed open-source MQ framework, and RabbitMQ is used in OpenStack. As for Pulsar, it is one of the fastest-growing MQ frameworks in recent years. The physical parameter settings of these MQ frameworks are the same as in [20]. We set 100 topics for each MQ framework and generate 200 VPCs by default. As shown in Fig. 7, VITA-SM performs better compared with NG and VITA-KM in all three MQ frameworks. Moreover, VITA-SM achieves lower control/MQ overhead, but results in higher message delay and higher total traffic amount on compute nodes than VITA-SA. That means, VITA-SM is more suitable for scenarios with limited processing capacity on the control plane or the MQ server, while VITA-SA is more suitable for scenarios with limited processing capacity on compute nodes. Note that, as shown in Figs. 7(c)-(d), RabbitMQ achieves the lowest total traffic amount, while achieves the smallest message delivery delay, compared with the other two frameworks. The reason is that RabbitMQ aims to obtain low message transmission delay, while the total throughput cannot be guaranteed. To save the space, we only conduct a detailed
performance comparison of all solutions when using Kafka in the following since it is the most widely used framework.

The second set of experiments observes the control/MQ overhead, average message delay, and total traffic amount of NG, VITA-SA, VITA-KM and VITA-SM by changing the number of available topics in the MQ server. The results are shown in Fig. 8, where the horizontal axis is the number of topics in the MQ server, ranging from 50 to 300. No matter how many topics there are in the MQ server, NG always achieves the worst performance compared with other solutions. For example, as shown in Fig. 8(b), given 200 topics, the average disk I/O utilization of VITA-SM, VITA-KM, VITA-SA and NG is 34%, 38%, 40% and 63%, respectively. That is, VITA-SM can reduce the average disk I/O utilization by about 10.5%, 15% and 46% compared with VITA-KM, VITA-SA and NG, respectively. We should note that, as shown in Fig. 8(c), when the number of topics exceeds 100, the average message delay will increase significantly as the number of topics increases. That means the MQ server can only support a limited number of topics. Thus, we should carry out message aggregation with a proper granularity.

The third set of experiments compares the control/MQ overhead, average message delay, and total traffic amount of NG, VITA-SA, VITA-KM and VITA-SM by changing the number of VPCs in the cloud. The results are shown in Fig. 9, where the number of VPCs ranges from 200 to 1,200. As the number of VPCs increases, all performance metrics (e.g., control overhead, MQ overhead, message delay and total traffic amount) increase for all algorithms. NG always achieves the worst performance compared with the other three VITA-based solutions. For example, when the number of VPCs reaches 1000, the average message delay of NG, VITA-SA, VITA-KM and VITA-SM is 54ms, 29ms, 45ms and 34ms. That means, VITA-SM can reduce the average message delay by 37% and 24.4% compared with NG and VITA-KM, respectively. VITA-based solutions are more efficient compared with NG, since southbound messages usually have VPC attributes, and VITA-based solutions aggregate messages with VPC as the granularity.

From these experimental results, we can draw some conclusions. First, as shown in Fig. 7, VITA-SA performs better in all three MQ frameworks compared with NG and VITA-KM, and achieves similar performance compared with VITA-SA. Second, Fig. 8 illustrates that the MQ server can only support a limited number of topics. Thus, we have to aggregate messages with a proper granularity. Third, the performance of NG lags behind all three VITA-based solutions for all metrics (e.g., control/MQ overhead, message delay and traffic amount on compute nodes). Fourth, our proposed VITA-SM performs better than VITA-KM, especially in the metrics of message delay and total traffic amount, which shows efficiency of our proposed message aggregation algorithm. Fifth, our proposed VITA-SM and VITA-SA algorithms have different application scenarios. If the control/MQ overhead become the network bottleneck, VITA-SM is a better choice compared with VITA-SA. Conversely, if resources on compute nodes are the network bottleneck, VITA-SA is a better choice compared with VITA-SM.

V. CONCLUSION

In this paper, we give the system overview of VITA and formulate the VSMD problem for minimizing the total amount of messages received by compute nodes. We propose a submodular-based algorithm for this problem and analyze its approximation performance. We further consider how to extend this scheme for more scenarios. Both the simulation and experimental results show high efficiency of our proposed VITA system.

ACKNOWLEDGEMENT

The corresponding authors of this paper are Gongming Zhao and Hongli Xu. We would also like to thank Xuwei Yang, Eric Li, James Chung for their contributions to this work. This article was supported in part by the National Science Foundation of China (NSFC) under Grants 61822210 and 62102392, and in part by the National Science Foundation of Jiangsu Province under Grant BK20210121.
REFERENCES


