B-Scale: Bottleneck-aware VNF Scaling and Flow Routing in Edge Clouds

Chen Chen*, Lars Nagel*, Lin Cui † and Fung Po Tso*
*Department of Computer Science, Loughborough University, UK
†Department of Computer Science, Jinan University, Guangzhou, China
Email: c.chen@lboro.ac.uk; l.nagel@lboro.ac.uk; tcuilin@jnu.edu.cn; p.tso@lboro.ac.uk

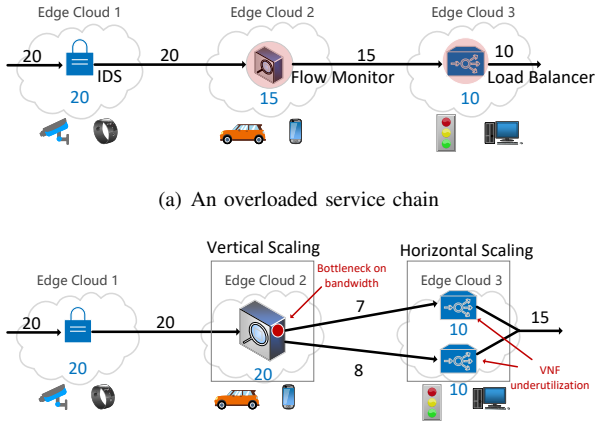
***Abstract*— With the ever-growing demand for low-latency network applications, edge computing emerges as a new paradigm that provides computation and storage resources in close proximity to end-users. Many research efforts have resorted to network function virtualization, wherein network applications are provisioned as service function chains at edge clouds. However, due to the traffic dynamics and limited resource capacity at the network edge, how to efficiently embed service chains with latency optimization and resource efficiency remains as a challenging problem.**

As most existing research efforts largely overlook the bottlenecked resources of VNFs in the VNF scaling, we seek a more realistic approach to provisioning VNF instances across multiple edge clouds. Also, given the limited resources at the edge, it is of significant importance to improve the VNF utilization rate. Specifically, we formulate the VNF scaling problem as an integer linear programming (ILP) problem, aiming to minimize the end-to-end latency for service function chains. To solve this problem, we devise a novel bottleneck-aware algorithm that manages the number and deployment of newly created instances. After that, we propose an online algorithm for traffic steering to improve the utilization rates of VNF instances and avoid congestion on hotspot links. The proposed algorithm is shown to provide good performance by trace-driven simulation in real-world topologies.

I. Introduction

Recently, Network Function Virtualization (NFV) has become a widespread paradigm to dynamically embed Virtualized Network Functions (VNFs) (e.g., firewall, flow monitor), chain them by leveraging Service Function Chain (SFC). Meanwhile, edge computing uses the SFC technology to deliver a variety of applications with low latency for end-users, which benefits real-time applications such as self-driving cars and augmented reality [1], [2], [3].

Edge computing processes computation tasks in the proximity of end-users, which reduces the end-to-end latency, improves the user experience and cuts down expenditures for service providers [4], [5]. Since edge clouds have limited capacities, we consider the VNF scaling technique to better utilize the scarce and valuable resources in edge clouds. In VNF scaling, every VNF may consist of multiple instances which are VNF replicas hosted on virtual machines or containers. Each VNF instance accommodates a few incoming flows to avoid resource limitations incurred by CPU, memory or bandwidth resource constraints on a single instance [6]. Hence, in this work, we focus on an important problem: *How to optimize the end-to-end latency of service function chains while improving the resource utilization at edge clouds by using VNF scaling?*



(a) An overloaded service chain

(b) After scaling by using existing solutions

Fig. 1. Existing solutions of vertical and horizontal scaling

It is very challenging to address the above problem in edge computing. Geo-distributed edge clouds need to share computation and network resources as the traffic volume can significantly differ at different locations or points of time. For instance, the traffic demand is aggregated at offices during working hours, while social or gaming traffic is concentrated at homes in the evening. Moreover, since the resource capacity in edge clouds is limited, the bandwidth resources on the shortest path can be insufficient in peak hours. As a result, some VNF instances can be spawned on non-shortest paths which can incur higher latency. Also, VNF instances on non-shortest paths will have a lower utilization rate than VNFs on the shortest path as most traffic would favor the latter.

However, most existing approaches largely overlook the inherent nature of VNFs and edge clouds. Many of them such as [7], [8], [9] assume that VNF instances can be scaled without limitations which is not viable in reality. CPU, memory or bandwidth can easily become bottlenecks limiting the performance of a VNF [10], [11], [12]. Secondly, many works [13], [9] ignore VNF underutilization incurred by resource reservation which exacerbates the resource scarcity in edge clouds.

One can observe that VNFs fall into two categories, namely I/O-bound (e.g., bottlenecked by bandwidth) and non-I/O-bound. As illustrated in Figure 1(a), the flow monitor is I/O-bound with a processing capacity of 15. Since the incoming traffic rate is 20, the flow monitor becomes overloaded. Like in Figure 1(b), existing solutions vertically scale up the

flow monitor by increasing the capacity to 20. However, the output rate of the flow monitor is still 15 because the flow monitor is bounded by bandwidth. In other words, scaling up the processing capacity by adding more CPU and memory cannot improve the network performance of I/O-bound VNFs. Similarly, the load balancer is also overloaded as the incoming traffic rate is greater than its processing capacity. Existing horizontal solutions would scale the load balancer by creating two instances with a processing capacity of 10. Nevertheless, the incoming traffic rates for both instances are 7 and 8, respectively. Therefore, horizontal scaling can result in resource overprovisioning and VNF underutilization.

In this paper, we aim to overcome the limitations of existing approaches by modeling the problem and devising an algorithm that takes the challenges into account. Our main contributions are:

- In the context of edge computing, we formulate a VNF scaling and traffic routing problem as an integer linear programming (ILP) problem with the aim to minimize the end-to-end latency.
- To solve this problem, we propose B-Scale, an online algorithm, which applies different scaling techniques where the type of scaling depends on the VNF category as well as the current traffic situation. Our solution also steers the network traffic by using the k shortest path algorithm between VNF instances to take advantage of non-shortest paths and avoid congestion on hotspot links.
- The solution is evaluated in a trace-driven simulation using real-world datasets. We demonstrate that B-Scale achieves near-optimal performance and improves the mean VNF utilization rate by 15%.

To the best of our knowledge, this is the first paper that considers the VNF category for VNF scaling in edge clouds.

The remainder of the paper is organized as follows. Section II summarizes the related works. Section III introduces the system model. Section IV proposes the optimization problem. Section V presents the proposed algorithm. Section VI evaluates the algorithm and conclusions are drawn in Section VII.

II. Related Work

Many research efforts have focused on horizontal scaling, aiming to accommodate the time-varying network traffic. FlexNFV [8] leverages horizontal scaling to distribute the traffic demand among multiple SF instances. Luo *et al.* [7] also investigate the VNF scaling problem by leveraging an online horizontal scaling approach. This approach ensures sufficient resources for the time-varying traffic demand. FastScale [13] proposes a chain-wide scaling approach by considering the downstream VNFs. Similarly, there have been some efforts on vertical scaling approaches for the VNF scaling problem. Qu *et al.* [14] propose a traffic parameter learning method that leverages change point detection and Gaussian process regression to study traffic parameters for every time slot. Luo *et al.* [15] resort to a deep-learning-based approach for predicting the traffic pattern over time. Trace-driven experiments verify that the proposed algorithm could timely achieve competitive system costs. Also, a few research efforts consider hybrid scaling approaches to avoid limitations of vertical or horizontal scaling. Fei *et al.* [16] propose a hybrid scaling approach to handle traffic fluctuations. By estimating the incoming traffic demand, this work creates new instances for overloaded network functions and routes the traffic. Gouareb *et al.* [9] propose two algorithms to minimize the edge cloud latency which leverage horizontal and vertical scaling, respectively. He *et al.* [6] propose a chain-aware scaling approach that computes the scaling sequence of SFs for service chains. Huang *et al.* [17] leverage a federated reinforcement learning method for SFC placement with fast convergence. Some research efforts leverage machine learning approaches to predict the traffic demand such as [18]. ENSC [11] applies priority management methods for VNF scaling.

Edge computing is characterized by the rapidly varying traffic load of multiple VNFs and the resource scarcity. However, little attention has been paid to the dynamic resource allocation of edge clouds with respect to the intrinsic nature of VNFs. Moreover, most existing works use the shortest path algorithm to route traffic between VNF instances which could incur congestion on hotspot links.

III. System Model

A. Physical Network and Service Chain Model

We use a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to model the network in which $\mathcal{V}$ and $\mathcal{E}$ denote the edge clouds and the inter-cloud links, respectively. Every edge cloud is a resource pool that consists of multiple NFV servers. If $e = (u, v) \in \mathcal{E}$, then the edge clouds u, $v \in \mathcal{V}$ are connected by the inter-cloud link e. By $C_v(t)$ we denote the remaining processing capacity of edge cloud v at time slot t, and by $C^{bw}_{uv}(t)$ the remaining bandwidth capacity of inter-cloud link $(u, v) \in \mathcal{E}$ at time slot t.

We use a 4-tuple $r = [S, T, \mathcal{N}, F]$ to represent an SFC request, where S and T are the ingress and egress edge cloud, respectively. $\mathcal{N} = n_1, n_2, ..., n_i$ is the set of VNFs, and F is the set of traffic flows. Every SFC request has multiple traffic flows $f \in F$. We use $\lambda_f(t)$ to denote the traffic rate of flow f. Recall that there can be multiple VNF instances for every VNF, which we refer to as VNF replicas or instances. We use $n^1_i, n^2_i, ..., n^j_i$ to denote the instances of VNF n_i.

B. Queuing Theory

Queuing theory and queuing models provide a good tool for analyzing the network delay. Since both VNF instances and network links can be modeled as a single server [9], we use the $M/M/1$ model to describe the queues at VNF instances and links.

The average delay $D(t)$ in the system consists of two parts: the average service time $\frac{1}{\mu(t)}$ and the average waiting time $W(t)$ [19].

$$D(t) = \frac{1}{\mu(t)} + W(t) \tag{1}$$

where $\mu(t)$ is the processing capacity of the system.

Let $\lambda(t)$ be the traffic rate at time slot t which is the total traffic rate of all flows in the system. Then, we apply the Little's Theorem [19] to Equation 1. The average delay in the system can be computed as follows.

$$D(t) = \frac{1}{\mu(t) - \lambda(t)} \tag{2}$$

TABLE I: Symbols and Variables

Symbols and Variables	Description
Physical Network	
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Physical network graph.
$C_v(t)$	Remaining processing capacity in the edge cloud v.
$C_{uv}^{bw}(t)$	Remaining bandwidth capacity in the inter-cloud link uv.
SFC Graph	
S	Ingress node of the SFC request.
T	Egress node of the SFC request.
n_i	VNF n_i in the SFC request.
n_i^j	jth instance of the VNF n_i.
$c_{n_i^j}(t)$	Amount of processing capacity required by the VNF instance n_i^j.
$b_{fuv}(t)$	Traffic rate of flow f in link uv.
Parameters	
$\lambda_{n_i}(t)$	Total traffic rate at VNF n_i.
$\mu_{n_i}(t)$	Total processing capacity of VNF n_i.
$\lambda_{n_i^j}(t)$	Traffic rate at VNF instance n_i^j.
$\mu_{n_i^j}(t)$	Processing capacity of VNF instance n_i^j.
Binary Variables	
$x_{fv}^{n_i^j}(t)$	Indicator variable equals 1 if VNF instance n_i^j of flow f is hosted on edge cloud v.
$y_{fuv}^{n_i^j n_{i+1}^j}(t)$	Indicator variable equals 1 if flow f traverses the virtual link $n_i^j n_{i+1}^j$ which is mapped to physical link uv.

C. VNF Scaling and Flow routing

Vertical scaling adds or reduces physical resources (e.g., CPU and memory) to an existing virtual machine or container. Horizontal scaling creates or removes VNF instances to process the network traffic.

In vertical scaling, the new processing capacity $\mu_{n_i}^{new}(t)$ required for VNF n_i is computed as follows.

$$\mu_{n_i}^{new}(t) = \begin{cases} 0 & \text{if } \lambda_{n_i}(t) \leq \lambda_{n_i}(t-1) \\ \mu_{n_i}(t) - \mu_{n_i}(t-1) & \text{otherwise.} \end{cases} \tag{3}$$

where $\lambda_{n_i}(t)$ and $\lambda_{n_i}(t-1)$ denote the total traffic rate at time slots t and $t-1$, respectively. $\mu_{n_i}(t)$ and $\mu_{n_i}(t-1)$ are the total processing capacity of VNF n_i at time slot t and $t-1$, respectively.

In horizontal scaling, the required number of new VNF instances $z_{n_i}^{new}(t)$ for VNF n_i is computed as follows.

$$z_{n_i}^{new}(t) = \begin{cases} 0 & \text{if } \lambda_{n_i}(t) \leq \lambda_{n_i}(t-1) \\ \lceil \frac{\mu_{n_i}(t) - \mu_{n_i}(t-1)}{\mu_{n_i}^{max}} \rceil & \text{otherwise.} \end{cases} \tag{4}$$

where $\mu_{n_i}^{max}$ represents the maximum processing capacity for one instance.

IV. Problem Description

A. Delay at VNF instances in edge clouds

According to Little's Theorem [19], we define the average delay at VNF instances in Equation (5), where $\mu_{n_i^j}(t)$ and $\lambda_{n_i^j}(t)$ are the processing capacity and the traffic rate at VNF instance n_i^j, respectively.

$$D_{n_i^j}(t) = \frac{1}{\mu_{n_i^j}(t) - \lambda_{n_i^j}(t)} \tag{5}$$

B. Link delay in inter-cloud links

Similarly, we define the average delay in inter-cloud links as follows. Let $C_{uv}(t)$ denote the processing capacity of the link uv and $B_{uv}(t)$ represents the total traffic rate in the link uv.

$$L_{uv}(t) = \frac{1}{C_{uv}(t) - B_{uv}(t)}, \quad \forall (u,v) \in \mathcal{E} \tag{6}$$

C. Objective Function

We formulate the ILP problem by considering delays both in edge clouds and inter-cloud links. We approximate these delays by using the average delays. All symbols and variables are listed in Table I.

$$\begin{aligned} min \quad & \sum_{f \in \mathcal{F}} \sum_{v \in \mathcal{V}} \sum_{n_i^j \in \mathcal{N}} x_{fv}^{n_i^j}(t) D_{n_i^j}(t) \\ & + \sum_{f \in \mathcal{F}} \sum_{uv \in \mathcal{E}} \sum_{n_i^j, n_{i+1}^j \in \mathcal{N}} y_{fuv}^{n_i^j n_{i+1}^j}(t) L_{uv}(t) \end{aligned} \tag{7}$$

where indicator variable $x_{fv}^{n_i^j}(t)$ equals 1 if flow f traverses the VNF instance n_i^j which is hosted on edge cloud v. Similarly, indicator variable $y_{fuv}^{n_i^j n_{i+1}^j}(t)$ equals 1 if flow f traverses the virtual link $n_i^j n_{i+1}^j$ which is mapped to the link uv.

$$\sum_{n_i^j \in n_i} \sum_{v \in V} x_{fv}^{n_i^j}(t) = 1, \quad \forall n_i \in \mathcal{N}, \quad \forall f \in \mathcal{F} \tag{8}$$

$$\sum_{f \in \mathcal{F}} \sum_{n_i^j \in \mathcal{N}} x_{fv}^{n_i^j}(t) c_{n_i^j}(t) \leq C_v(t), \quad \forall v \in \mathcal{V} \tag{9}$$

$$\sum_{f \in \mathcal{F}} \sum_{n_i^j, n_{i+1}^j \in \mathcal{N}} y_{fuv}^{n_i^j n_{i+1}^j}(t) b_{fuv}(t) \leq C_{uv}^{bw}(t) \tag{10}$$

Equation (8) ensures that each flow traverses only one instance of each VNF. Equation (9) ensures that the required processing capacities must not exceed the remaining processing capacity $C_v(t)$. Equation (10) ensures that the traffic rate must not exceed the remaining bandwidth capacity $C_{uv}^{bw}(t)$.

V. B-Scale Algorithm

The VNF scaling problem has been proved to be NP-hard in many existing works [9], [16]. We show that the Multiple Knapsack Problem (MKP) can be reduced to our problem. Let each VNF instance n_i^j denote an item m with the size $m_i^j.req$, where the size $m_i^j.req = c_{n_i^j}(t)$. Let each edge cloud v denote a knapsack k with a capacity $k_v.cap = C_v(t)$. Assume that the profit of assigning flows to each VNF instance is the negative of the flow delays. Then, our problem becomes finding VNF placement for each flow that maximizes the total profit. In other words, the MKP problem is a special case of our problem. As the MKP problem has been proven to be NP-hard [20], our problem is NP-hard.

A. Online Bottleneck-Aware VNF Scaling Algorithm

As shown in Algorithm 1, for each VNF n_i, B-Scale first observes the actual traffic rate at n_i. If the total traffic rate $\lambda_{n_i}(t)$ decreases compared with time slot $t-1$, the algorithm checks the existing VNF instances. If the traffic rate $\lambda_{n_i^j}(t)$

Algorithm 1: Online bottleneck-aware VNF Scaling Algorithm for Latency Minimization

Input : Request r, Graph $\mathcal{G}$;
for t = *1,2,...,T* **do**
  for n_i *in* $\mathcal{N}$ **do**
    Observe actual traffic rate $\lambda_{n_i}(t)$ at VNF n_i;
    if $\lambda_{n_i}(t) \leq \lambda_{n_i}(t-1)$ **then**
      for n_i^j *in* n_i **do**
        if $\lambda_{n_i^j}(t) == 0$ **then**
          release n_i^j;
        else if $\lambda_{n_i^j}(t) \leq \mu_{n_i^j}(t)$ **then**
          Scale down $\mu_{n_i^j}(t)$ to $\lambda_{n_i^j}(t)$;
        end
      end
    else
      Call **Place()** to deploy VNFs;
      Call **Traffic steer()** to route traffic;
    end
  end
  Update residual processing $C_v(t)$ and bandwidth capacity $C_{uv}^{bw}(t)$, $\forall v \in \mathcal{V}$, $\forall uv \in \mathcal{E}$;
end

Algorithm 2: Place()

Compute the set P of k shortest paths between S and T;
for *shortest path* p *in* P **do**
  if n_i *is non-I/O-bound* **then**
    Compute new required processing capacity $\mu_{n_i}^{new}(t)$;
    Choose the first-fit edge cloud v in p;
    Perform vertical scaling acc. to $\mu_{n_i}^{new}(t)$;
    if *Vertical scaling failed* **then**
      Compute the number of new instances $z_{n_i}^{new}(t)$;
      foreach *newly created instance* n_i^j **do**
        Choose first-fit edge cloud v in p;
        Creating new instance horizontally;
      end
    end
  end
  else if n_i *is I/O-bound* **then**
    Compute the number of instance $z_{n_i}^{new}(t)$;
    foreach *newly created instance* n_i^j **do**
      Choose first-fit edge cloud v in p;
      Creating new instance horizontally;
    end
  end
end

at an instance equals 0, the algorithm removes the instance. If the traffic rate $\lambda_{n_i^j}(t)$ at an instance is smaller than its processing capacity $\mu_{n_i^j}(t)$, the algorithm scales down the processing capacity $\mu_{n_i^j}(t)$ to the traffic rate $\lambda_{n_i^j}(t)$. If the total traffic rate $\lambda_{n_i}(t)$ increases compared with time slot $t-1$, the algorithm uses Algorithm 2 to deploy new VNF instances. If all VNF instances are successfully placed, the algorithm invokes Algorithm 3 to route the traffic for service chains. Finally, the algorithm updates the processing and bandwidth capacities for the system.

B. New Instance Placement in Edge Clouds

Algorithm 2 first computes the k shortest paths between ingress and egress pairs. After that, the algorithm examines the VNF category. If the VNF is non-I/O-bound, the algorithm performs vertical scaling (scale up) based on the required new processing capacity $\mu_{n_i}^{new}(t)$. The first-fit edge cloud on the path p is used to host this instance. In our case, the term "first-fit" refers to the first edge cloud with sufficient processing capacity to host the instance. If vertical scaling fails due to the limited capacity in the edge cloud, the algorithm performs horizontal scaling instead. The algorithm first computes the number of required new instances $z_{n_i}^{new}(t)$. Then, the algorithm assigns the instances to the first-fit edge cloud on the path p. In contrast, if the VNF is I/O-bound, the algorithm performs only horizontal scaling because vertical scaling cannot improve the performance.

Algorithm 3: Traffic steer()

for *new flow* f **do**
  foreach *VNF* n_i **do**
    Find the least used new instance n_i^j;
    Check the residual processing capacity of n_i^j;
    Assign f to n_i^j;
  end
  foreach *VNF* n_i **do**
    for p' *in* k *shortest paths* P' *between* n_i^j *and next-hop instance* n_{i+1}^j **do**
      if *Check bandwidth succeeds* **then**
        Steer the flow along p';
        Update resource capacities;
        break;
      end
    end
  end
end

C. Traffic Steer Algorithm

We employ Algorithm 3 to compute a valid path for new flows. Different from most existing solutions that employ the shortest path algorithm, we employ a k shortest path algorithm to take advantage of spare links on non-shortest paths. To avoid the complex state migration problem, we assume that we only steer new coming packets ("new flow") to these newly created instances at time slot t. In other words, the routing of existing packets remains unchanged.

At time slot t, there are multiple newly created instances for each VNF n_i. For every new flow f in the service chain, the algorithm assigns it to the least used instance n_i^j. Then, between every VNF instance n_i^j and the next-hop VNF instance n_{i+1}^j, the algorithm creates multiple candidate paths from the k shortest algorithm. After that, the algorithm selects the first-fit path with sufficient bandwidth to route the traffic.

VI. Performance Evaluation

A. Simulation Setup

We conducted extensive simulations with Network Simulator 3 (NS3) on a server with 105 GB RAM and an Intel(R) Xeon(R) E5645 processor with 24 cores. The SFC requests are derived from .pcap files of the CAIDA traces [21]. Every SFC consists a chain of 3 VNFs randomly selected from Table II. The

TABLE II: Processing time and processing capacity of VNFs

VNF	Processing time	Max. Processing capacity	I/O-bound
Firewall	120μs	400Mbps	Yes
IDS	160μs	600Mbps	No
Caching	83μs	580Mbps	No
Flow monitor	200μs	550Mbps	Yes
Load balancer	647.5μs	500Mbps	No

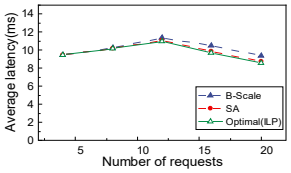


Fig. 2. Average latency

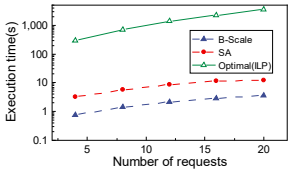


Fig. 3. Execution time

processing time and capacity of VNFs is derived from [16]. The VNF category is derived from related works [10], [11]. We proportionally scale down the VNF processing capacity by a factor of 10 to adapt to the resource capacities of the system. The packet length is assumed to be 512 bytes on average. The traffic change ratio is derived from [22]. The processing capacity is 400 Mpbs for each edge cloud and the link bandwidth is 400 Mbps.

We adopt the simulated annealing (SA) algorithm [23] to approximate the global optimal solution as it is an effective searching technique for combinational optimization problems [24]. We first generate a random solution in SA algorithm. Then, we generate a new solution by changing the placement of one SFC. After that, SA decides to accept or reject the new solution. SA only accepts a solution if all capacity constraints are fulfilled. If the end-to-end latency is not improved after one hundred iterations or the temperature is cooled down to 1, SA accepts the current solution and terminates the iteration. We do not implement other approaches for comparison from existing solutions as they do not consider the VNF category in the VNF scaling problem.

B. B-Scale in Topology Agis

First, we compare B-Scale and SA with the optimum achieved by the ILP solver Gurobi [25]. Since the ILP solver cannot find solutions for large-scale networks and hundreds of requests in a reasonable time, we only implement the ILP solver for a small topology named Agis with 25 nodes and 30 links from Internet Topology Zoo.

Figure 2 shows that SA approximates the global optimal solution well as SA only leads to 1% more latency than the optimum on average. B-Scale achieves 4% more latency compared with the optimum. However, Figure 3 shows that the ILP solver spends over 3400 seconds to find solutions for only 20 requests. In contrast, the execution time of B-Scale and SA is only 12 seconds in the worst case. Overall, the results suggest that SA can closely approximate the global optimal solution and hence we use SA as a baseline to benchmark the performance of B-Scale in a larger topology in the following sections.

C. Offline B-Scale in Topology TW

As it includes a variety of geo-distributed networks, we chose a US topology named TW which is composed of 76 nodes and 123 links from the Internet Topology Zoo. We first examine the performance of B-Scale in an offline scenario which means that all SFC requests start to run at the beginning and only leave the system when the simulation is ended. We conducted two sets of experiments that simulated light and heavy workloads. Light means that the overall traffic demand is low with respect to the processing and bandwidth capacities in the system, heavy means that it is high. The maximum

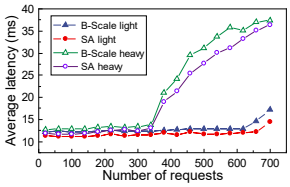


Fig. 4. Average latency

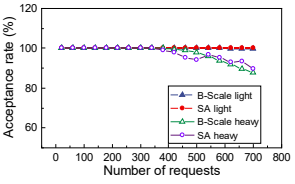


Fig. 5. Acceptance rate

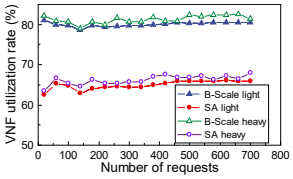


Fig. 6. VNF utilization rate

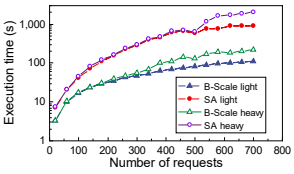


Fig. 7. Execution time

traffic demands of light and heavy workloads are about 5400 and 9500 Mb/s, respectively. We test the B-Scale algorithm against a different number of SFC requests ranging from 20 to 700 with the step size 40. A larger number of requests results in a larger amount of network traffic, which requires more VNF instances to process.

Fig. 4 shows that B-Scale closely matches the optimal solution approximated by SA for both light and heavy workloads. When the workload is light, the average latency of both algorithms only increases slightly because there are sufficient resources on the shortest paths between ingress and egress pairs. When the workload is heavy, the average latency of both algorithms rises sharply when the number of SFC requests becomes greater than 340 as bandwidth resources gradually become insufficient. On average, the latency of B-Scale is about 9% higher than that of SA. The result indicates that B-Scale achieves a similar performance compared with SA. Fig. 5 shows the acceptance rate which is the ratio between successfully deployed SFC requests and total SFC requests. For light workloads, B-Scale and SA both achieve almost 100 % acceptance rate. For heavy workloads, B-Scale is only outperformed by 2-3% in a few cases compared with SA. Overall, B-Scale achieves an acceptance rate close to the one of SA. Fig. 6 illustrates the mean VNF utilization rate. For light and heavy workloads, B-Scale outperforms SA by about 15% in most cases. This result indicates that B-Scale efficiently utilizes the computation resources and hence mitigates resource underutilization. Fig. 7 shows the total execution time of both algorithms for heavy and light workloads. B-Scale significantly reduces the execution time by up to 90%. The execution time of B-Scale for each request ranges from 0.16 to 0.3 seconds on average. This result suggests that B-Scale works in a timely manner and hence is suitable for online VNF placement.

D. Online B-Scale in Topology TW

We also evaluate the performance of both algorithms in an online scenario where SFCs arrive and leave at different times. We use the timestamp derived from the CAIDA traces [21] as the start time of the flow. The average lifecycle of every flow is 1000 seconds. The simulation simulates 18000 seconds. Figure 8 shows the distribution of end-to-

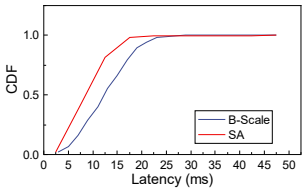


Fig. 8. CDF of latency

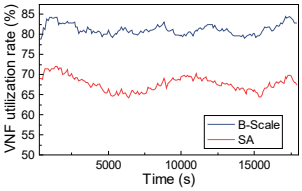


Fig. 9. VNF utilization rate

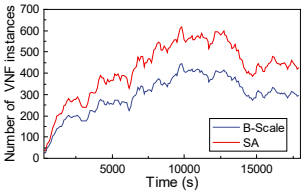


Fig. 10. Number of instances

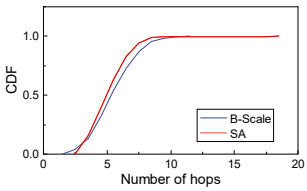


Fig. 11. CDF of number of hops

end latency. The 99^{th} percentile latency of B-Scale is 25 ms while that of SA is 22.5 ms. This result suggests that B-Scale achieves a comparable performance compared with SA. Since most SFCs are deployed within 25 ms, the proposed B-Scale algorithm effectively reduces the end-to-end latency and avoids congestion. Figure 9 shows that B-Scale remarkably improves the VNF utilization rate by about 15% in most cases which implies that B-Scale efficiently utilizes the computation resources and mitigates VNF underutilization. These findings imply that B-Scale is suitable for SFC orchestration in edge clouds as the resource capacity in edge clouds are smaller than that in public clouds. Figure 10 shows the number of VNF instances over time. B-Scale significantly reduces the number of instances by up to 33% which suggests that B-Scale efficiently utilizes the system resources. The rationale is that B-Scale uses vertical and horizontal scaling based on different types of VNF which efficiently reduces the number of VNF instances. Finally, Figure 11 provides insights into the number of hops for flows. The 99^{th} percentile number of hops is 10.5 for B-Scale while that of SA is 9.5. This result suggests that B-Scale approximates the number of hops achieved by SA.

Overall, B-Scale achieves similar end-to-end latency and significantly improves the VNF utilization rate compared with SA. Also, B-Scale reduces the execution time remarkably which means that it scales well especially when the problem size increases.

VII. Conclusion

In this paper, we investigate the VNF scaling problem in edge clouds. We formulate an ILP problem that minimizes the end-to-end latency for service chains. When embedding new instances based on the VNF category, an online scaling approach is adopted for reducing the end-to-end latency and improving the resource utilization. Extensive experimental results show that the proposed algorithm can achieve comparable latency and improve the VNF utilization rate by 15% compared with SA. Meanwhile, the proposed algorithm can reduce the execution time by up to 90% at the cost of light performance degradation in the acceptance rate.

References

[1] Y. Liu, X. Shang, and Y. Yang. Joint sfc deployment and resource management in heterogeneous edge for latency minimization. *IEEE Transactions on Parallel and Distributed Systems*, 32(8):2131–2143, 2021.

[2] X. Li, Z. Zhou, C. Zhu, L. Shu, and J. Zhou. Online reconfiguration of latency-aware iot services in edge networks. *IEEE Internet of Things Journal*, pages 1–1, 2021.

[3] B. Dai, F. Xu, Y. Cao, and Y. Xu. Hybrid sensing data fusion of cooperative perception for autonomous driving with augmented vehicular reality. *IEEE Systems Journal*, 15(1):1413–1422, 2021.

[4] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu. Delay-aware virtual network function placement and routing in edge clouds. *IEEE Transactions on Mobile Computing*, 20(2):445–459, 2021.

[5] Y. Gao, L. Liu, X. Zheng, C. Zhang, and H. Ma. Federated sensing: Edge-cloud elastic collaborative learning for intelligent sensing. *IEEE Internet of Things Journal*, 8(14):11100–11111, 2021.

[6] L. He, L. Li, and Y. Liu. Towards chain-aware scaling detection in nfv with reinforcement learning. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–10, 2021.

[7] Z. Luo and C. Wu. An online algorithm for vnf service chain scaling in datacenters. *IEEE/ACM Transactions on Networking*, 28(3):1061–1073, 2020.

[8] X. Fei, F. Liu, H. Jin, and B. Li. Flexnfv: Flexible network service chaining with dynamic scaling. *IEEE Network*, 34(4):203–209, 2020.

[9] R. Gouareb, V. Friderikos, and A. Aghvami. Virtual network functions routing and placement for edge cloud latency minimization. *IEEE Journal on Selected Areas in Communications*, 36(10):2346–2357, 2018.

[10] S. Van Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester. Profile-based resource allocation for virtualized network functions. *IEEE Transactions on Network and Service Management*, 16(4):1374–1388, 2019.

[11] H. Yu, J. Yang, C. Fung, R. Boutaba, and Y. Zhuang. Ensc: Multi-resource hybrid scaling for elastic network service chain in clouds. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 34–41, 2018.

[12] W. Hajji, T. A. L. Genez, F. P. Tso, L. Cui, and I. Phillips. Dynamic network function chain composition for mitigating network latency. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 00316–00321, 2018.

[13] X. Chen, Y. Chen, Q. Huang, H. Zhou, D. Zhang, C. Wu, and J. Xing. Fastscale: Fast scaling out of network functions. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 436–442, 2020.

[14] K. Qu, W. Zhuang, X. Shen, X. Li, and J. Rao. Dynamic resource scaling for vnf over nonstationary traffic: A learning approach. *IEEE Transactions on Cognitive Communications and Networking*, 7(2):648–662, 2021.

[15] Z. Luo, C. Wu, Z. Li, and W. Zhou. Scaling geo-distributed network function chains: A prediction and learning framework. *IEEE Journal on Selected Areas in Communications*, 37(8):1838–1850, 2019.

[16] X. Fei, F. Liu, H. Xu, and H. Jin. Adaptive vnf scaling and flow routing with proactive demand prediction. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 486–494, 2018.

[17] H. Huang, C. Zeng, Y. Zhao, G. Min, Y. Zhu, W. Miao, and J. Hu. Scalable orchestration of service function chains in nfv-enabled networks: A federated reinforcement learning approach. *IEEE Journal on Selected Areas in Communications*, 39(8):2558–2571, 2021.

[18] Y. Yao, S. Guo, P. Li, G. Liu, and Y. Zeng. Forecasting assisted vnf scaling in nfv-enabled networks. *Computer Networks*, 168:107040, 2020.

[19] D. P. Bertsekas and R. G. Gallager. *Data Networks, 2nd ed.* Upper Saddle River, NJ, USA: Prentice-Hall, 1992.

[20] H. Kellerer, U. Pferschy, and D. Pisinger. Knapsack problems. *Berlin, Germany: Springer*, 2004.

[21] CAIDA. The caida ucsd passive-2019.

[22] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou. Traffic aware placement of interdependent nfv middleboxes. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.

[23] P. Laarhoven and E. Aarts. Simulated annealing. In *Simulated Annealing: Theory and Applications*. Springer Netherlands, 1987.

[24] D. Bhamare, A. Erbad, R. Jain, M. Zolanvari, and M. Samaka. Efficient virtual network function placement strategies for cloud radio access networks. *Computer Communications*, 127:50–60, 2018.

[25] Gurobi. Gurobi optimizer reference manual, 2021.