An NFV-based Scheduling and Flexible Deployment Scheme for RaaS Functions in Cloud Data Centers

Hsueh-Wen Tseng
Computer Science and Engineering
National Chung-Hsing University
Taiwan, R.O.C.
hwtseng@nchu.edu.tw

Ting-Ting Yang
Computer Science and Engineering
National Defense University
Taiwan, R.O.C.
bestwishytt@ccit.ndu.edu.tw

Pei-Shan Chen
Computer Science and Engineering
National Chung-Hsing University
Taiwan, R.O.C.
memory03656@gmail.com

***Abstract*—Recommender systems appear on many commercial websites because E-Commerce and social websites are popular. However, many businesses do not have enough capacity to develop their recommender systems, which must be outsourced by cloud-based service providers because the amount and complexity of data increase. Recently, network function virtualization (NFV) has been proposed to use virtualization technology to replace dedicated network equipment for software network functions. Virtual network functions (VNFs) run on commodity servers or standard physical machines. In this paper, an NFV-based scheduling and flexible deployment scheme (SFDS) is proposed to consider the characteristics of recommendation-as-a-service (RaaS), including the popularity and execution order of the functions. According to the popularity of the functions, a dynamic deployment algorithm is able to flexibly create or remove the VNFs on the virtual nodes. Finally, the simulation results show a shorter completion time. SFDS enhances the resource utilization of the system and has a better successful reception rate.**

***Index Terms*—Recommender System, RaaS, Network Function Virtualization, Scheduling Algorithm, Flexible Deployment.**

I. Introduction

In recent years, due to the development of electronic commerce and the popularization of social networks, recommender systems have attracted great attention in academia and industry. Most current recommender systems propose different recommendation algorithms for the recommended accuracy, different products, or special behavior patterns [1]. Some studies aimed to develop different recommender systems to recommend different types of items to users [2] [3] [4]. However, due to the increasing amount of operational data and computational complexity, merchants do not have enough resources to develop their own systems. They must outsource or rent resources from recommender service providers [5] [6]. Therefore, recommender service providers provide the cloud platform in the data center to fulfill the diverse requirements of various vendors. In the future, there will be more and more applications built on the cloud platform [7].

In the past, hardware-specific devices can only handle specific or customized functions. It constantly increases hardware demand and spends more time and construction costs. The concept of network function virtualization (NFV) technology uses virtualization to enable network functionality that is implemented in a software manner and executes on a general commodity device to replace the hardware-specific device. In addition, it is possible to deploy and build customized functions more quickly and flexibly. Thus, NFV greatly reduces operating cost and improves the efficiency of network function deployment.

A network service is divided into multiple functions using a software-based approach to deploy network functions on virtual machines (i.e., virtual network function (VNF)). Thus, it can establish or remove VNFs flexibly, and thus NFV can provide new services quickly. We use NFV to provide many kinds of recommendation services which are produced by vary recommender systems on the cloud platform. As a result, cloud service providers can conveniently manage and flexibly deploy recommendation functions. In addition, they can quickly adjust and create new functions in response to requests from different users, and then generate recommended results for users.

In general, recommender systems include the following components: data collection and processing, recommender model, recommendation post-processing, online modules, and user interface [8]. Most of the recommended results will be generated according to the execution order of the above components. In the recommender service, the execution order of the functions is limited. If a function of one service waits for functions of the other services to be completed, then subsequent functions of this service cannot be performed. Thus, there are unexpected delays, and the recommended results cannot be generated in a timely manner for the user. At the same time, the user experience will reduce, even causing the loss of customers and affecting the revenue of operators.

Service providers deploy many types of recommender system on the cloud platform. Every recommender system is customized based on the requests of different users. Due to the wide variety of recommender systems, it has become quite difficult to manage recommender systems. To solve the above-mentioned problems, we observe some features in the recommender services. First, the functions of a recommender service must be performed in a specific order. Second, different recommender systems provide customized algorithms to different customers and the recommended items are not exactly the same. As a result, the functions of different recommender systems are not used interactively. Based on the features, we propose an NFV-based scheduling and flexible deployment scheme (SFDS) in this paper.

To match the applications of the actual recommender system, we consider the priority of the service, the popularity of each function, and the relationship between the components of the recommender system. For dynamic deployment, an evaluation function is proposed to determine the use of the function. Our method improves the performance of the service execution, increases the operational effectiveness of the overall system, and the resource utilization of the cloud platform.

The remainder of this paper is organized as follows. Section II reviews related work on the NFV scheduling and VNF placement scheme. Section III describes the system model and the scheme proposed in the paper in detail. Section IV shows the simulation results. Finally, Section V is the conclusion.

II. RELATED WORK

In order to design the scheduling and dynamic deployment mechanism for recommender systems, this section investigates the literature on NFV scheduling and VNF placement issues. [9] mentioned that the real-time scheduling problem was defined as the flexible job-shop scheduling problem (FJSP). It defined network function mapping and scheduling (NFMS) to solve the online scheduling problem in NFV. This problem is divided into two parts: the first part is how to select suitable nodes to execute VNFs, and the other part is the scheduling schemes for VNFs.

[10] discussed two types of delay in the NFV scheduling problem: transmission delay and processing delay. Taking into account the transmission delay between virtual nodes, the authors used a genetic algorithm that can schedule functions and dynamically adjust the bandwidth of the virtual link to reduce the completion time of the service. However, this study uses traditional job-shop scheduling to solve the NFV scheduling problem, but does not support real-time service. [11] proposed the network service chaining (NSC) algorithm to find the best VNF to process the function of the network service. Additionally, the paper considered CPU, memory, network bandwidth, and cost to define the evaluation metric and used the genetic algorithm to find the best solution.

Some researches had discussed that VNFs are deployed on physical or virtual nodes. With limited resources and lower cost, they used efficient placement strategies to minimize management costs or the number of VNFs allocated. [12] used MILP to solve the VNF placement problem. MILP not only achieves effective resource utilization, avoids network congestion and server overload, but also meets the QoS requirements. [13] used integer linear programming to formulate the VNF placement problem and considered the optimal use of operational cost and resources to decide the number of VNFs allocated without violating the SLA.

[14] focused on the deployment of VNFs in the data center network (DCN). To minimize east-west traffic in the DCN, the authors discussed the number of VNFs, the allocation of VNF nodes, and how to separate network traffic. [15] proposed a Markov decision process (MDP) control model to dynamically assess the cost of building VNF and removing VNF and the cost of processing delay. Therefore, they can determine the number of VNFs to be deployed at the lowest cost. Furthermore, when the processing delay cost suddenly increases, the decision maker will adjust the VNF traffic to achieve load balance.

[16] focused on a task scheduling algorithm based on a hierarchical network architecture and a general topology. The scheduling algorithm considered the transmission time for uploading and downloading and energy consumption. However, [16] did not consider the processing time and the waiting time in the queue. Therefore, the scheduling algorithm cannot meet the real-time constraints of the recommendation system. Furthermore, [16] did not consider the deployment problem and the scheduling problem of the VNFs in the VMs. In [17]- [20], the learning-based scheduling algorithm, such as reinforcement learning (RL), provides an approach to long-term cost minimization, with the ability to capture inherent patterns in network dynamics and make intelligent decisions accordingly. These works focused on delay-sensitive services.

As mentioned in the previous description, we propose an NFV-based scheduling and dynamic deployment mechanism. Previous studies have discussed NFV scheduling and VNF placement issues, respectively, whereas we use the effective VNF placement and dynamic adjustment mechanism to improve the efficiency of NFV scheduling. However, the previous methods focus on the network function, but the RaaS function of the recommender system is more complicated than the network function. Furthermore, previous NFV scheduling methods dynamically adjust the placement of VNFs in virtual nodes, but do not consider the frequency of use of the functions or the characteristic of the service request (i.e., the relationship between components). Thus, the system performance will degrade in the long term and even cannot handle the user request. Our proposed scheme will be introduced in detail in Section III.

III. NFV-BASED SCHEDULING AND FLEXIBLE DEPLOYMENT SCHEME (SFDS)

A. System Architecture

To resolve the problem mentioned in the previous section, we propose an NFV-based scheduling and flexible deployment scheme (SFDS) that consists of three phases: initialization, scheduling, and dynamical deployment. The scheme can be applied to the same constraint or to the same scenario for other services. We show the overall system architecture and the algorithm flow chart as shown in Fig. 1. We assume that VMs have to be built on servers on a cloud platform to start the initialization phase, as shown in step 1. After deploying all VNFs to appropriate VMs, the system starts waiting for the user request (i.e., the service request). After that, the scheduling scheme (i.e., RaaS function mapping and scheduling; RFMS) allocates the functions of the service request to available VMs to be executed in the system, as shown in step 2. Finally, we use an evaluation function to calculate the current usage of each function and determine whether the functions must be adjusted for the deployment of VNF on VMs, as shown in step 3.

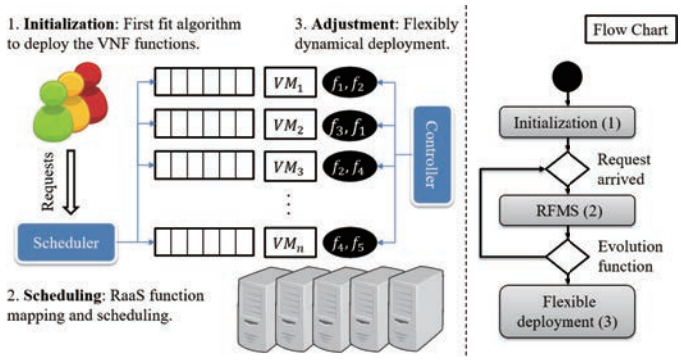

Fig. 1: System architecture

We refer to the problem of network function mapping and scheduling (NFMS) according to the paper [9]. Thus, we establish a system model and design a scheduling algorithm. Let $G(V, E)$ be a virtual network, $V = \{1, \ldots, n\}$ represents that there are n virtual nodes (VMs) that are built on physical nodes; $E(v_i \rightarrow v_j)$ represents the distance between node i and j. Thus, the number of full connection links is $n \times (n-1)$. In this paper, we consider the request of the user to use multiple resources. However, the most important resources for recommender services are CPU and memory [14]; therefore, we define the available CPU and memory of node j as $cpu(j)$ and $mem(j)$, respectively.

Assume that s_k denotes the k_{th} recommender service to reach the system. Each recommender service s_k is made up of m VNFs that are executed in a specific order. Each virtual node processes at most one function at the same time [10]. Furthermore, the deadline for the recommendation service (s_k) is indicated by D_k. Additionally, we consider the transmission delay in the scheduling algorithm. C_e^{sd} denotes the available transmission bandwidth of link e_d^s (from node s to node d), and R_{k_i} is the request for the bandwidth of the function i in the service k. Thus, the transmission delay is R_{k_i}/C_e^{sd} and π_e^{sd} is the time at which the link e_d^s completes the transmission.

In recommender services, recommender functions can be classified into five components [8]. The number of components does not affect the performance of the proposed scheduling algorithm. We redefine the set of functions as $F_p = \{f_{p,1}, f_{p,2}, f_{p,3}, \ldots, f_{p,q}\}$ $(1 \leq p \leq 5)$, where $f_{p,q}$ denotes the q_{th} function of the p_{th} component; $cpu(f_{p,q})$ and $mem(f_{p,q})$ are the CPU requirement and the memory requirement of the function $f_{p,q}$, respectively. The buffer size used by the function $f_{p,q}$ is $\delta_{p,q}$, and the buffer size available from node j is B_j.

$v(f_{p,q}) \subseteq V$ is a set of nodes available for the execution of the function $f_{p,q}$. The processing time of each function $f_{p,q}$ at node j is $\rho_{f_{p,q},j}$. Furthermore, $\rho_{f_{p,q},j} > 0$, where $j \in v(f_{p,q})$ and $1 \leq j \leq n$; $B_{f_{p,q},j}$ is a binary variable used to determine whether node j can perform the function $f_{p,q}$ or not, as shown in (1).

$$B_{f_{p,q},j} = \begin{cases} 1, \text{the node } j \text{ for executing function } f_{p,q} \\ 0, \text{otherwise} \end{cases} \quad (1)$$

In the algorithm, to select nodes to perform each function in a convenient way, the completion time of the function $f_{p,q}$ in the service is denoted by $t_{p,q}$, and the completion time of the last function in the service must be less than or equal to D_k; the start time of the function $f_{p,q}$ is $t_{p,q}^s$. Finally, π_j is the expected completion time of node j. In other words, it is the completion time of the last job in the queue of node j. We refer to [9] to propose a greedy algorithm to find an optimal solution. The following is a detailed description for the NFV-based scheduling and flexible deployment scheme (SFDS).

B. NFV-based Scheduling and Flexible Deployment Scheme

In [9], authors proposed three greedy algorithms, which consider the shortest execution time in the virtual node (greedy fast processing, GFP), the queue has the earliest completion time (greedy best availability, GBA), and the queue has the most available buffer time (greedy least loaded, GLL), respectively. However, in this paper, we consider that the function has the shortest waiting time (greedy least waiting time, GLWT) in RFMS. In addition, we also consider the transmission delay between functions.

Algorithm 1 is RaaS function mapping and scheduling algorithm (RFMS). When the service request arrives at the system, it first finds nodes that can handle the functions of the service request (line 3); nodes have to ensure that the functions are completed before the deadline and there are enough buffer sizes in nodes. If there is no node that can meet the constraints, the allocation is stopped, and the error times are recorded by the controller.

After finding the set of nodes ($V^{'}$), the waiting time of the function on each node (w_{k_id}) will be calculated, including the time to wait for transmission ($e_{k\_i}^{w\_sd}$), the transmission time ($e_{k\_i}^{r\_sd}$), and the time to wait for processing ($v_{k\_id}^{w}$). The system then assigns functions to nodes that have the shortest waiting time (lines 5-21). When the function is completed at a node s, if other functions transmit on the link, the result must wait for other functions to complete the transmission on the link. Thus, it generates the waiting time ($e_{k\_i}^{w\_sd}$); the transmission time ($e_{k\_i}^{r\_sd}$) is defined by which the data are actually transmitted over the link. After the function is transmitted to a node d, if there are other functions waiting to be processed on node d, the function should wait for the previous functions to finish. Thus, it generates the waiting time for processing ($v_{k\_id}^{w}$).

In Algorithm 1, lines 7-12, it computes the transmission time from a source node (s) to a destination node (d). Then we find a node that has the shortest waiting time, and then assign the function to the node (lines 14-20). In line 14, it calculates the time that is the previous function of node s to transmit to node d ($e_{k\_i-1}^{sd}$). In line 15, it computes the waiting time for processing the function on the node d. In line 16, it determines whether the waiting time (from line 15) is smaller than the shortest waiting time. Finally, in lines 17-18, it determines the time which is the function to perform the process and transmission. In lines 22-24, it computes the completion time of the function i and allocates the function f_i to the queue on the node d. When the last function is completed, the result will

Algorithm 1 RaaS Function Mapping and Scheduling

1: //One service arrives at the system
2: **for** each function $f_i \in k$ **do**
3: Find the set of available node $V^{'}$.
4: //Select the node d with the least w_{k_id}.
5: **for** each node $d \in V^{'}$ **do**
6: //Select available link to transmit.
7: **if** link e connected the source s and the destination d is available **then**
8: //Calculate waiting time to transmit.
9: $e^{a\_sd}_{k\_i-1} = t_{k\_i-1}$;
10: $e^{w\_sd}_{k\_i-1} = \max(\pi^{sd}_e, e^{a\_sd}_{k\_i-1}) - e^{a\_sd}_{k\_i-1}$;
11: $e^{r\_sd}_{k\_i-1} = R_{k\_i-1}/C^{sd}_e$;//Transmission time
12: **end if**
13: //Calculate waiting time to process.
14: $t^a_{k\_i} = e^{sd}_{k\_i-1} = e^{w\_sd}_{k\_i-1} + e^{r\_sd}_{k\_i-1}$;
15: $v^w_{k\_id}$=max$(\pi_d, t^a_{k_i}) - t^a_{k_i}$;
16: **if** $w_{k\_id} > (e^{w\_sd}_{k\_i-1} + e^{r\_sd}_{k\_i-1} + v^w_{k\_id})$ **then**
17: $e^{s\_sd}_{k\_i-1} = \max(\pi^{sd}_e, t_{k\_i-1})$;
18: $t^s_{k\_id} = \max(\pi_d, t^a_{k\_i})$;
19: $w_{k\_id} = e^{w\_sd}_{k\_i-1} + e^{r\_sd}_{k\_i-1} + v^w_{k\_id}$;
20: **end if**
21: **end for**
22: $t_{k\_i} = e^{sd}_{k\_i-1} + v^w_{k\_id} + \rho_{k\_id}$;
23: Map f_i onto the node d.
24: Update the queue of the node d.
25: **if** f_i is the final function **then**
26: Find one e of the links connected to the destination d.
27: Calculate transmission time R_{k_i}/C^{sd}_e.
28: **end if**
29: **end for**

be sent to the destination and also generates the transmission delay (lines 25-28).

To improve the performance of the recommender services on the cloud platform, the algorithm considers five components of the recommended services. In the initialization phase, the functions of the first component in the service are preallocated to nodes, and then the remaining functions are allocated to others. In order to reduce the transmission delay, it finds that the front and rear components of the same service are allocated to nodes which have the shortest transmission time to the node with the current component. In addition, when the system is looking for the node, it judges whether the resources of the node meet the requirements of the function. In er systems, CPU and memory resources are the most important [21]. Finally, if the system finds the available node, it allocates the function to the node directly to improve the efficiency of the allocation.

In Algorithm 2, it assigns VNFs to nodes. In lines 2-6, it first deploys functions of the first component. In lines 3-4, if there is only one node m to process the first function, it chooses the node m to process the first function. in line 6, if there are nodes to process the first function, it finds that node m has the shortest distance to node n to which the previous function is assigned, and the link (node m to node n) must meet the bandwidth requirement of the link. Then, it deploys the remaining functions of the second to the fifth components to nodes as shown in lines 11-20. In line 12, it finds a set of nodes ($V^{'}$) that can perform the previous function ($f_{p-1,q}$) and the next function ($f_{p+1,q}$) for the function ($f_{p,q}$). In lines 13-19, it finds a node s from $V^{'}$, and s has the shortest distance to the current node and meets the resource requirements. Finally, it deploys the function on the node s.

Algorithm 2 First-fit Algorithm for VNF Placement

1: //Place the first component F_1
2: **for** each function $f_{1,q} \in F_1$ **do**
3: **if** $q == 1$ **then**
4: //Find the first available node m.
5: **else**
6: Find the node m with the shortest path $e(n \rightarrow m) \in E$ and $C_e \geq R_{f_{1,q}}$.
7: **end if**
8: $n = m$;
9: **end for**
10: //Place residual component set $F_p(2 \leq p \leq 5)$
11: **for** function $f_{p,q} \in eachF_p$ **do**
12: Find the set of node $V^{'}$ placed $f_{p-1,q}$ or $f_{p+1,q}$.
13: **for** each $r \in V^{'}$ **do**
14: Find the node s with the shortest path $e(r \rightarrow s) \in E$ and $C_e \geq R_{f_{p,q}}$.
15: **if** $cpu(s) > cpu(f_{p,q})\&\&mem(s) > mem(f_{p,q})$ **then**
16: Allocate $f_{p,q}$ to the node s.
17: $break$
18: **end if**
19: **end for**
20: **end for**

Finally, based on the use of functions, VNFs are dynamically deployed on nodes. Therefore, the evaluation function is used to calculate x_i, and α is a critical value (threshold) of the evaluation function. When we use Algorithm 2 to complete the deployment of VNF, Algorithm 3 is performed. Algorithm 3 is to evaluate the use of each function and determine whether the deployment of VNFs at the nodes needs to be adjusted.

In Algorithm 3, line 2, it removes the VNF that has not been used for a long time. It uses the evaluation function to calculate the x_i value as shown in line 4. In line 5, it determines whether x_i is greater than α or not. In lines 7-22, it creates a new VNF on a node in response to system requirements. When $x_i > \alpha$, it finds nodes that meet the resource requirements of the system. In lines 9-14, it determines whether there is an available link or not. If there is an available link, it continues with subsequent processes. Then, in line 15, it calculates the average waiting time (AWT) and the number of functions assigned on node j (N_j). In lines 16-20, it finds the node k with the smallest AWT and the smallest N_j. Finally, in line 23, it allocates the function to the node k.

Algorithm 3 Flexibly Dynamic Deployment

//Dynamical VNF remove
(i, k);
//Dynamical VNF creation
x_i;
if $x_i > \alpha$//VNFCreate(); **then**
  $minAWT = 0$;
  for each node $j \in V$ **do**
    //Has available links to transmit.
    for each link e connected with the node j **do**
      if $C_e^{sj} > R_i$ **then**
        hasLink=true;
      end if
    end for
    if $cpu(j) > cpu(f_i)$ and $mem(j) > mem(f_i)$ and $hasLink$ **then**
      Calculate $AWT_j = \frac{\sum_{i=1}^{m} B_{ij} \times w_{ij}}{\sum_{j=1}^{n} \gamma_{ij}}$; $N_j = \sum_{i=1}^{m} B_{ij}$.
      if $minAWT > AWT_j$ and $minCount > N_j$ **then**
        $k = j$;
        $minAWT = AWT_j$;
        $minCount = N_j$;
      end if
    end if
  end for
  Allocate function i on the node k.
end if

Last but not least, the mechanism for removing VNFs on virtual nodes is described. When each function is established at the node, the variable $y_{i,j}$ has a constant value C. The value is reduced by one when there is no function i to allocate at node j; If the function i is allocated at node j, the value of $y_{i,j}$ maintains the same value. The purpose is to monitor VNFs that have not been used for a long time and can be removed immediately. Furthermore, the mechanism also prevents VNF from occupying too much resources and resulting in poor utilization and resource waste.

IV. Simulation Results

TABLE I: Simulation parameters

Parameter	Range or Value
The number of nodes	50
Buffer capacity for each node	[75, 100]
The number of processed functions on each node	[1, 7]
Processing time of each function	[15, 30]
Buffer demand for each function	[7.5, 10]
The number of functions for each service	[5, 10]
The deadline of services	[5000, 10000]
The bandwidth demand of each function	[1, 4] Mbps
The maximal bandwidth of each link	8 Mbps

In Section IV, our RFMS is compared with GBA [9]. We show the performance of using the initialization (INIT) and dynamic adjustment (FDD) methods. We observe the differences before and after INIT and FDD are used. Next, we describe the simulation environment and the simulation results in sequence. We use a discrete event simulator to perform the simulation and use Java to implement the simulation [9] [22]. In the simulation environment, we assume that the service request arrival process forms the Poisson distribution with the average arrival rate λ. The average service arrival rate is set to every 5 time units per one service request. There are 2000 service requests to arrive at the simulation. The parameters are used as shown in Table I [9] [10]. In the simulation, the metrics are average completion time and successful reception rate. The average completion time is the average time to complete a service. Successful reception rate is defined as the number of successful services over the number of all services.

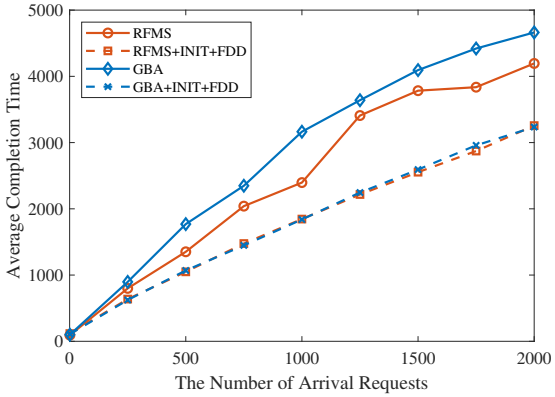

Fig. 2: Average completion time of RMFS and GBA

We consider the characteristics of the recommender service. There are 50 functions in the system. In INIT, it assigns the functions to the nodes according to the relationship between the components. The functions are classified into 5 components, and each component contains 10 functions. On the basis of the order of the components, one of the functions is randomly selected from each component to build the service. Next, we show the simulation results for using INIT and FDD.

Fig. 2 shows the average completion time of RFMS and GBA with/without INIT and FDD. In Fig. 2, we observe that INIT and FDD can effectively reduce the average completion time of the service. When the system is initialized, INIT strategically allocates the functions to the nodes. This is because INIT considers the relation between the components to effectively deploy the functions to nodes. On the other hand, after FDD is used, FDD dynamically adjusts the allocation of the functions on the nodes based on the use situation of the functions. Thus, FDD can effectively reduce the average completion time of services.

Fig. 3 shows the successful reception rate of services. The original scheduling algorithms (i.e., RFMS and GBA) have a lower successful reception rate without using INIT and FDD. This is because the RFMS and GBA randomly allocate the functions to the nodes. Therefore, when the number of arrival services increases, the successful reception rate decreases. After INIT and FDD are used, the successful reception rate of

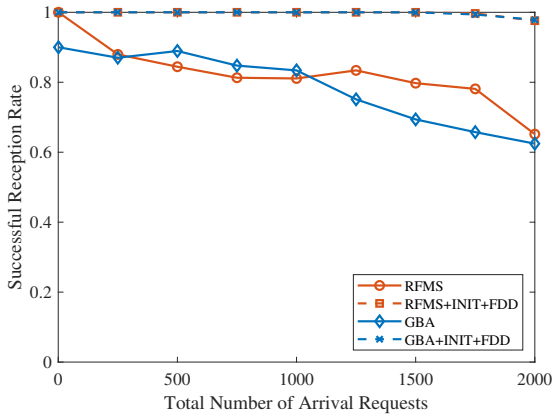

Fig. 3: Successful reception rate of RFMS and GBA

services almost keeps the best value because we effectively use the initial allocation and the flexible management mechanism to improve the average completion time of services.

In summary, our method can effectively reduce the response time of recommender services and enhance the efficiency of the overall system. Additionally, whether or not our algorithm (RFMS) is applied to recommender services, FDD effectively manages and dynamically adjusts the placement of the VNFs to greatly improve the processing time of services. However, dynamic adjustment requires additional time and cost to build and remove the VNF.

V. CONCLUSION

This paper considers the characteristics of the recommender services in the cloud environment, such as the functions must be executed in order, and the functions of different recommender services are not used interactively. Thus, we propose a scheme based on NFV-based scheduling and flexible deployment. The scheme effectively reduces the completion time of the recommender services. In addition, the effective deployment and adjustment mechanism can increase the performance of recommender services and the convenience of service providers. The simulation results show that our proposed algorithm has better processing efficiency.

ACKNOWLEDGMENT

The authors would like to thank the Ministry of Science and Technology, Republic of China (Taiwan), for financially supporting this research under Contract MOST 108-2221-E-005-019-MY3, MOST 109-2222-E-606-001-MY2 and MOST 110-2224-E-005-001.

REFERENCES

[1] W. Li, D. Hu, and J. Luo, "A Personalized Service Recommendation Algorithm for Service Functionality," Intl. Conference on Advanced Cloud and Big Data, pp. 275-281, 2014.

[2] G. Jung, T. Mukherjee, S. Kunde, H. Kim, N. Sharma and F. Goetz, "CloudAdvisor: A Recommendation-as-a-Service Platform for Cloud Configuration and Pricing," IEEE Ninth World Congress on Services, pp. 456-463, 2013.

[3] A. Umanets, A. Ferreira and N. Leite, "GuideMe–A tourist guide with a recommender system and social interaction," Elsevier Procedia Technology, pp. 407-414, 2014.

[4] T. Ku, H. Won, and H. Choi, "Service recommendation system for big data analysis," Intl. Conference on Information Networking (ICOIN), pp. 317-320, 2016.

[5] D. Ben-Shimon, L. Rokach, G. Shani, and B. Shapira, "Anytime Algorithms for Recommendation Service Providers," ACM Trans. Intell. Syst. Technol. 7, 3, Article 43 (April), 26 pages, 2016.

[6] F. Ricci, L. Rokach, and B. Shapira, "Recommender Systems: Introduction and Challenges," in Recommender Systems Handbook. Springer, Boston, MA, 2015.

[7] X. Dai, X. Wang, and N. Liu, "Optimal Scheduling of Data-Intensive Applications in Cloud-Based Video Distribution Services," in IEEE Trans. on Circuits and Systems for Video Technology, vol. 27, no. 1, pp. 73-83, Jan. 2017.

[8] Maya Hristakeva and Kris Jack, "A Practical Guide to Building Recommender Systems," 2016, [Online]. Available: https://buildingrecommenders.wordpress.com/

[9] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), 2015, pp. 1-9.

[10] L. Qu, C. Assi, and K. Shaban, "Delay-Aware Scheduling and Resource Optimization with Network Function Virtualization," in IEEE Trans. on Communications, vol. 64, no. 9, pp. 3746-3758, Sept. 2016.

[11] T. Kim, S. Kim, K. Lee and S. Park, "A QoS Assured Network Service Chaining Algorithm in Network Function Virtualization Architecture," 15th IEEE/ACM Intl. Symposium on Cluster, Cloud, and Grid Computing, pp. 1221-1224, 2015.

[12] F. Ben Jemaa, G. Pujolle, and M. Pariente, "QoS-Aware VNF Placement Optimization in Edge-Central Carrier Cloud Architecture," IEEE Global Communications Conference (GLOBECOM), pp. 1-7, 2016.

[13] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating Virtualized Network Functions," in IEEE Trans. on Network and Service Management, vol. 13, no. 4, pp. 725-739, Dec. 2016.

[14] P. Chi, Y. Huang, and C. Lei, "Efficient NFV deployment in data center networks," IEEE Intl. Conference on Communications (ICC), pp. 5290-5295, 2015.

[15] M. Shifrin, E. Biton, and O. Gurewitz, "Optimal control of VNF deployment and scheduling," IEEE Intl. Conference on the Science of Electrical Engineering (ICSEE), pp. 1-5, 2016.

[16] W. Zhang and Y. Wen, "Energy-Efficient Task Execution for Application as a General Topology in Mobile Cloud Computing," in IEEE Trans. on Cloud Computing, vol. 6, no. 3, pp. 708-719, 1 July-Sept. 2018.

[17] H. Li, K. Ota, and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," in IEEE Network, vol. 32, no. 1, pp. 96-101, Jan.-Feb. 2018.

[18] Z. Luo, C. Wu, Z. Li and W. Zhou, "Scaling Geo-Distributed Network Function Chains: A Prediction and Learning Framework," in IEEE Journal on Selected Areas in Communications, vol. 37, no. 8, pp. 1838-1850, Aug. 2019.

[19] J. Liu, H. Guo, J. Xiong, N. Kato, J. Zhang, and Y. Zhang, "Smart and Resilient EV Charging in SDN-Enhanced Vehicular Edge Computing Networks," in IEEE Journal on Selected Areas in Communications, vol. 38, no. 1, pp. 217-228, Jan. 2020.

[20] J. Li, W. Shi, N. Zhang, and X. Shen, "Delay-Aware VNF Scheduling: A Reinforcement Learning Approach With Variable Action Set," in IEEE Trans. on Cognitive Communications and Networking, vol. 7, no. 1, pp. 304-318, March 2021.

[21] Han, S. M., Hassan, M. M., Yoon, C. W., and Huh, E. N., "Efficient service recommendation system for cloud computing market," in Proceedings of the 2nd Intl. Conference on Interaction Sciences: Information Technology, Culture and Human, pp. 839-845, 2019.

[22] P. Bratley, B. L. Fox, and L. E. Schrage, "Writing a Discrete Event Simulation: Ten Easy Lessons," 2016, [Online]. Available: https://users.cs.northwestern.edu/ agupta/_projects/networking/QueueSimulation/mm1.html