Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 517409, 15 pages
http://dx.doi.org/10.1155/2015/517409

Research Article

Cost-Efficient Allocation of Additional Resources for the Service Placement Problem in Next-Generation Internet

Ding Ma,[1,2] M. Onderwater,[3,4] F. Wetzels,[3] G. J. Hoekstra,[3] R. D. van der Mei,[3,4] S. Bhulai,[4] and Lei Zhuang[1]

[1] *School of Information and Engineering, Zhengzhou University, Zhengzhou 450001, China*
[2] *College of Information Science and Engineering, Henan University of Technology, Zhengzhou 450001, China*
[3] *CWI, 1098 XG Amsterdam, Netherlands*
[4] *VU University Amsterdam, 1081 HV Amsterdam, Netherlands*

Correspondence should be addressed to Ding Ma; dma@gs.zzu.edu.cn

Received 24 March 2015; Revised 24 June 2015; Accepted 5 July 2015

Academic Editor: Fabio Tramontana

One of the major challenges in next-generation Internet is to allocate services to nodes in the network. This problem, known as the *service placement problem*, can be solved by layered graph approach. However, due to the existence of resource bottleneck, the requests are rejected from the beginning in the resource constrained network. In this paper we propose two iterative algorithms for efficient allocation of additional resources in order to improve the ratio of accepted service placement requests. To this end, we (1) introduce a new concept of sensitivity for each service node to locate the bottleneck node, (2) state the problem of allocating additional resources, and (3) use sensitivity to propose a simple iterative algorithm and an utilization-based iterative algorithm for efficient resource allocation. The performance of these two algorithms is evaluated by simulation experiments in a variety of parameter settings. The results show that the proposed algorithms increase request acceptance ratio significantly by allocating additional resources into the bottleneck node and links. The utilization-based iterative algorithm also decreases the long-term cost by making efficient use of additional resources.

1. Introduction

A huge achievement has been made in the Internet technology over the last four decades in supporting a wide array of distributed applications and in providing fundamental end-to-end communication connectivity. However, with the increase of the Internet's scale and scope of use, some inherent deficiencies of the Internet architecture are gradually exposed. Innovations are needed in the following aspects of the current Internet architecture, namely, mobility support, reliability and availability, and quality of service guarantees [1]. To realize these innovations, many promising network architectures have been designed for the next-generation Internet.

An important approach adopted by some next-generation Internet architectures is to move the data processing functions from the end-systems to the routers inside the core of network. In the context of such networks, the data processing function is considered as service (*in-network* service [2]), and the transcoding, encryption, flow control, multicast, and so forth are examples of the service [2]. This approach provides greater flexibility with its ability to compose the services along the data path to satisfy different communication requirements from the end-system applications [3, 4]. In such network architectures, a major challenge is to determine which nodes the services are placed on along the data path and determine the shortest path between these nodes. This problem is known as the *service placement problem* which is proven to be NP-hard when considering constraints of the network resources [5].

In the service placement problem, to establish an end-to-end connection, the sequence of services that represents the application functionality and the required network resources need to be specified in advance in the end user's request. The data from one end system to another needs to be routed to pass through the nodes where certain services specified in the sequence of services are available while the network resources are sufficient on these nodes. Apparently, the service placement problem is quite different from the traditional routing problem in which the data always follow the shortest path. Furthermore, in the service-centric network architectures (e.g., *Network Service Architecture* (*NSA*)), the service controller performs the mapping algorithm to determine where to place which services [2, 3].

The layered graph algorithm, with low computational complexity, is an efficient solution to the service placement problem when the resource is unlimited. Yet it shows limitations in finding valid end-to-end connections due to the existence of resource bottleneck in the resource constrained network [5]. In this paper, we focus on how to locate the bottleneck based on the layered graph algorithm and to increase the resource capacity of the bottleneck to improve the performance of the resource constrained network. To this end, we first introduce a new concept, *sensitivity*, to represent the impact that a single service node can have on the performance of the entire network in terms of average ratio of accepted service placement requests. To compute the sensitivity of a service node, we remove this service node from the service network while maintaining the network working to measure the decrease rate in average request acceptance ratio. According to the value of sensitivities, we locate the bottleneck node which is corresponding to the service node with the greatest sensitivity value. We then propose two sensitivity-based iterative algorithms, called SI-AAR and UI-AAR, for solving the problem of allocating additional resources into the bottleneck node and its adjacent links to increase the average request acceptance ratio. Both of them first compute the sensitivities for each node in the network. SI-AAR then provides a simple way to iteratively increase both the CPU capacity and bandwidth capacity by the same increase rate. UI-AAR can iteratively supplement additional resources selectively based on resource utilization ratio. The results from our experiments show that our sensitivity-based iterative algorithms increase request acceptance ratio significantly by allocating additional resources into the bottleneck node and links. UI-AAR outperforms SI-AAR in terms of the long-term average cost and the amount of allocated additional resources by making more efficient use of additional resources.

The rest of the paper is organized as follows. In Section 2, we first overview the related work briefly. Then we state the service placement problem in Section 3. In Section 4, we discuss the concept of the sensitivity and state the problem of allocating additional resources. Section 5 describes the method of computing the sensitivity and two sensitivity-based iterative algorithms, SI-AAR and UI-AAR. Section 6 presents experimental results in a variety of parameter settings. In Section 7, we conclude this paper and outline future research directions.

2. Related Work

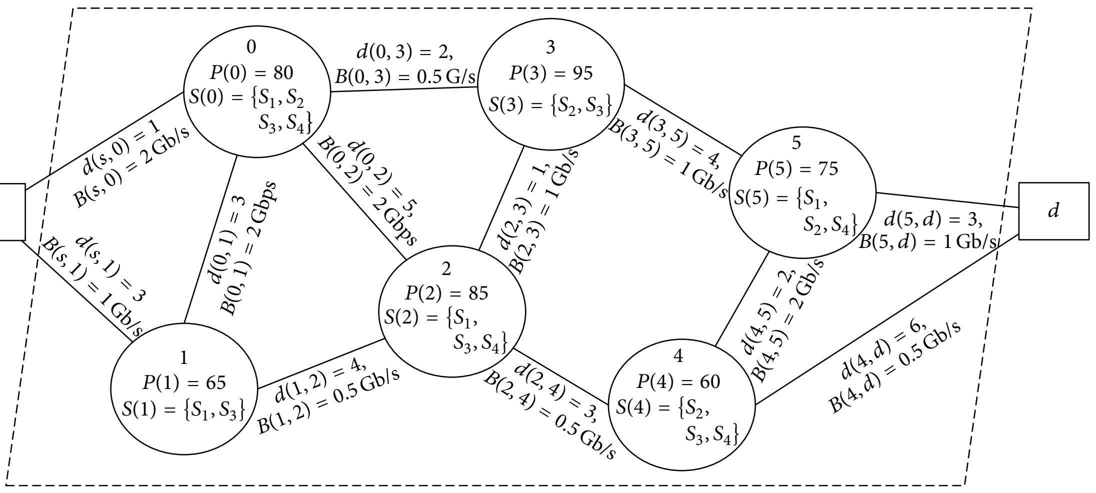
Some network architectures have been designed to support *in-network* services and provide network functional composition. The *Service Integration Control and Optimization* (*SILO*) project considers building blocks of fine-grain functionality as services and combines services to accomplish highly configurable complex communication tasks [4, 6]. The NSA implements an abstraction for communication between end-systems by providing packet processing service inside the network [2, 3]. Recent routing architectures such as programmable routers [7, 8] and virtualized router platforms [9] have provided technical support for implementation of the above network architectures.

Related service placement problems have been discussed extensively in recent work. In programmable networks, (1) Choi et al. [5] presented a layered graph algorithm to solve the service placement problem. In this algorithm, a multiple-layer graph is constructed and Dijkstra's shortest path algorithm is applied on the layered graph to find the shortest path. Then they proposed a capacity tracking approach to prevent the overuse of resources when considering capacity constraints. But they did not consider the impact that the bottleneck has on the performance of entire network in terms of request blocking rate. (2) Huang et al. [10, 11] proposed a distributed solution to the service placement problem in resource unconstrained network. By introducing a service matrix, this distributed algorithm can determine the optimal or near-optimal routes for connection requests. The advantage of this distributed algorithm is that it is more suitable for large-scale networks.

In service overlay network, (1) Raman and Katz [12] also used the layered graph algorithm and focused on load balancing without considering the capacity constraints. By introducing a *least-inverse-available-capacity* (*LIAC*) metric to reassign the link cost in the layered graph, it is ensured that the links with lower load are preferred over the links with higher load. (2) Liang and Nahrstedt [13] presented a greedy heuristic algorithm to solve the service composition problem to find low-cost composition solutions. (3) Qian et al. [14] also used heuristic algorithm to establish composite services delivery path with lowest cost. In addition, they considered the changes of data size along the data path when choosing services hop-by-hop.

In cloud environment, Tran et al. [15] used recurrence method to solve the service composition problem. They discussed three exact algorithms for three different topologies: path, star, and tree.

To the best of our knowledge, this paper is the first proposal that studies the problem of bottleneck locating and utilizing in the service-centric network architecture. Here, we introduce several related research works in other network architectures. In the Internet, Hu et al. [16] presented a novel light-weight, single-end active probing tool (Pathneck) which allows end users to efficiently and accurately locate bottleneck. According to the location information of bottlenecks, they used multihoming and overlay routing to avoid bottlenecks. In virtualized network, (1) Butt et al. [17] presented a topology-aware measure method to detect the bottleneck nodes and links in the substrate network. And then they proposed a set of algorithms for reoptimizing and remapping initially rejected virtual network requests. (2) Based on the analysis on resource utilization, Fajjari et al. [18] found that the existence of bottlenecked substrate links is the main reason of most of the request rejections. Given



s
d
0
P(0) = 80
S(0) = {S1, S2, S3, S4}
1
P(1) = 65
S(1) = {S1, S3}
2
P(2) = 85
S(2) = {S1, S3, S4}
3
P(3) = 95
S(3) = {S2, S3}
4
P(4) = 60
S(4) = {S2, S3, S4}
5
P(5) = 75
S(5) = {S1, S2, S4}
d(s, 0) = 1, B(s, 0) = 2 Gb/s
d(s, 1) = 3, B(s, 1) = 1 Gb/s
d(0, 1) = 3, B(0, 1) = 2 Gbps
d(0, 2) = 5, B(0, 2) = 2 Gbps
d(0, 3) = 2, B(0, 3) = 0.5 G/s
d(1, 2) = 4, B(1, 2) = 0.5 Gb/s
d(2, 3) = 1, B(2, 3) = 1 Gb/s
d(2, 4) = 3, B(2, 4) = 0.5 Gb/s
d(3, 5) = 4, B(3, 5) = 1 Gb/s
d(4, 5) = 2, B(4, 5) = 2 Gb/s
d(5, d) = 3, B(5, d) = 1 Gb/s
d(4, d) = 6, B(4, d) = 0.5 Gb/s

FIGURE 1: Service Network.

that they proposed a reactive and iterative algorithm for remapping the rejected request through migration of nodes and its bottlenecked attached links. However, none of them consider adding additional resources to the bottleneck node and links to improve the performance of entire network.

3. Service Placement Problem

In the network architecture where data processing functions can be implemented inside the network, we term this physical network as *service network*, the node in this service network as *service node*, and the link in this service network as *service link*. Then we state the service placement problem formally as follows.

3.1. Service Network. We model the service network as a weighted undirected graph and denote it by $G = (N, L)$, where N is the set of service nodes and L is the set of service links. The number of service nodes and the number service links are denoted by $|N|$ and $|L|$, respectively. Each service node $n_i \in N$ is associated with the CPU capacity weight value $P(n_i)$, available service set $S(n_i) = \{S_\tau \mid$ service S_τ is available on service node $n_i\}$, and service S_τ's processing time weight value $d_{S_\tau}(n_i)$ on service node n_i for service node resources. Each service link $l(i, j) \in L$ between service node n_i and n_j is associated with the link bandwidth weight value $B(l)$ which denotes the total amount of bandwidth capacity and its link delay weight value $d_l \equiv d(i, j)$ for service link resources.

An example of the service network topology is shown in Figure 1.

3.2. Request. An end user's request for end-to-end customized composite services can be represented as a set including six elements and denoted by $R = (n_s, n_d, t_a, t_d, b, S^R)$. Here n_s is the source node, n_d is the destination node, t_a is the arrival time, t_d is the duration time, b is the required bandwidth capacity, and S^R is the required service set which is composed of service number sn and an ordered list of services sl. Each service $\text{sl}_\jmath \in \text{sl}$, where $\jmath$ represents the index of services in the ordered list, is associated with the CPU capacity requirement weight value $p(\text{sl}_\jmath)$. For example, a request $R = (s, d, 90, 1050, 200\,(\text{Mb/s}), \{4, (S_3 \rightarrow S_1 \rightarrow S_4 \rightarrow S_2)\})$ while $p(\text{sl}_1) = 5$, $p(\text{sl}_2) = 10$, $p(\text{sl}_3) = 10$, $p(\text{sl}_4) = 5$.

3.3. Service Path. Given the end user's request R and the service network G, an *end-to-end service path* is such a path from the source service node n_s to the destination service node n_d, and all required services in the service list sl should be processed in sequence along this path.

3.3.1. Service-to-Node Mapping. Each service from the ordered service list sl needs to be processed by a service node in the end-to-end service path by a mapping $\mathscr{M}_{\mathcal{S}} : \text{sl} \rightarrow N$ from services to service nodes such that, for all $\text{sl}_\jmath \in \text{sl}$,

$$\mathscr{M}_{\mathcal{S}}\left(\text{sl}_\jmath\right) \in N, \quad (1)$$

where (1) if $\mathscr{M}_{\mathcal{S}}(\text{sl}_\jmath) = \mathscr{M}_{\mathcal{S}}(\text{sl}_{\jmath'})$, $\jmath \neq \jmath'$, then $\text{sl}_\jmath$ is not necessarily equal to $\text{sl}_{\jmath'}$, which means multiple services can be performed on a single service node; (2) if $\mathscr{M}_{\mathcal{S}}(\text{sl}_\jmath) = \varnothing$, that indicates the service $\text{sl}_\jmath$ is not performed on any service node.

3.3.2. Service Path. Given the service-to-node mapping, the end-to-end service path is denoted by

$$\mathscr{P}^{e2e} = \left\{\left(n_{i_1}, M_{i_1}\right), \left(n_{i_2}, M_{i_2}\right), \ldots, \left(n_{i_m}, M_{i_m}\right)\right\}, \quad (2)$$

where n_{i_1} is the source node, $n_{i_1} = n_s$, n_{i_m} is the destination node, $n_{i_m} = n_d$, $n_{i_2}, n_{i_3}, \ldots, n_{i_{m-1}}$ are the service nodes, $l(i_k, i_{k+1})$ is the service link, the hops of $\mathscr{P}^{e2e}$ denoted by $\text{hops}(\mathscr{P}^{e2e})$ are equal to $m - 1$, and M_{i_k} is a service-to-node mapping set on service node n_{i_k} defined as

$$M_{i_k} = \left\{\left(\text{sl}_\jmath \longrightarrow n_{i_k}\right) \mid \mathscr{M}_S\left(\text{sl}_\jmath\right) = n_{i_k}\right\}, \quad (3)$$

where if $M_{i_k} = \varnothing$, that indicates no service is performed on service node n_{i_k}.

Table 1: Service processing time on service nodes.

Node	Service			
	S_1	S_2	S_3	S_4
0	$d_{S_1}(0) = 3$	$d_{S_2}(0) = 4$	$d_{S_3}(0) = 4$	$d_{S_4}(0) = 2$
1	$d_{S_1}(1) = 5$	—	$d_{S_3}(1) = 3$	—
2	$d_{S_1}(2) = 2$	—	$d_{S_3}(2) = 6$	$d_{S_4}(2) = 1$
3	—	$d_{S_2}(3) = 1$	$d_{S_3}(3) = 2$	—
4	—	$d_{S_2}(4) = 6$	$d_{S_3}(4) = 8$	$d_{S_4}(4) = 6$
5	$d_{S_1}(5) = 4$	$d_{S_2}(5) = 3$	—	$d_{S_4}(5) = 4$

To guarantee the validity of the service path, several requirements have to be met.

(1) All service nodes have sufficient CPU capacity for performing the mapped services such that, for $\forall n_{i_k} \in \mathscr{P}^{e2e}$,

$$P\left(n_{i_k}\right) \geq \sum_{\forall \mathrm{sl}_J \rightarrow n_{i_k} \in M_{i_k}} p\left(\mathrm{sl}_J\right), \quad (4)$$

where $\mathrm{sl}_J \rightarrow n_{i_k} \in M_{i_k}$ indicates that service sl_J is performed on service node n_{i_k}.

(2) All service links have sufficient link bandwidth such that, for $\forall l(i_k, i_{k+1}) \in \mathscr{P}^{e2e}$,

$$B\left(l\left(i_k, i_{k+1}\right)\right) \geq b * at, \quad (5)$$

where the service link $l(i_k, i_{k+1})$ appears at times in $\mathscr{P}^{e2e}$.

Given the end user's request R outlined above, service network (depicted in Figure 1), and the service processing time on service nodes (depicted in Table 1), the service paths $\mathscr{P}_1^{e2e} = \{(s, \varnothing), (0, \{\mathrm{sl}_1 \rightarrow 0\}), (2, \{\mathrm{sl}_2 \rightarrow 2\}), (4, \{\mathrm{sl}_3 \rightarrow 4\}), (5, \{\mathrm{sl}_4 \rightarrow 5\}), (d, \varnothing)\}$ and $\mathscr{P}_2^{e2e} = \{(s, \varnothing), (1, \varnothing), (2, \{\mathrm{sl}_1 \rightarrow 2, \mathrm{sl}_2 \rightarrow 2\}), (0, \{\mathrm{sl}_3 \rightarrow 0\}), (3, \{\mathrm{sl}_4 \rightarrow 3\}), (5, \varnothing), (d, \varnothing)\}$ are both valid for request R. In $\mathscr{P}_1^{e2e}$, each service is performed on one service node; in $\mathscr{P}_2^{e2e}$, services S_3 (sl_1) and S_1 (sl_2) are performed on service node 2, and no service is performed on service nodes 1 and 5.

3.3.3. Objective. The delay of an end-to-end service path $\mathbb{D}(\mathscr{P}^{e2e})$ is defined as the summation of service processing time on service nodes and communication delay on service links along the service path

$$\mathbb{D}\left(\mathscr{P}^{e2e}\right) = \sum_{k=1}^{m-1} d\left(i_k, i_{k+1}\right) + \sum_{k=2}^{m-1} \sum_{\mathrm{sl}_J \rightarrow n_{i_k} \in M_{i_k}} d_{\mathrm{sl}_J}\left(n_{i_k}\right), \quad (6)$$

where $n_s = n_{i_1}, n_d = n_{i_m}$, and m is the number of service nodes in $\mathscr{P}^{e2e}$. The objective of service placement problem in this paper is to find a least delay service path from all valid $\mathscr{P}^{e2e}$.

Due to the finite nature of network resources, capacity constraints are the crucial considerations for solving the service placement problem. When an end-to-end connection request arrives, the service network has to determine whether to accept the request or not according to its specification. If the request is accepted, the service network operator needs to place services on service nodes, allocate the CPU capacity on the corresponding service nodes, and link bandwidth on service links to establish the least delay end-to-end service path. Once the end user leaves, the service path is destroyed and the allocated resources are released.

In this paper, we make several assumptions as follows:

(1) We assume that requirements of resources and services specified in an end user's connection request do not change over the duration time of the connection.

(2) An end-to-end service path, which is established according to an end user's connection request, is fixed during the lifetime of this connection.

4. Problem of Allocating Additional Resources into Service Network Based on Sensitivity

The layered graph with capacity tracking algorithm is an efficient approach to solve the service placement problem. However, the layered graph algorithm cannot perform well when the capacity of network resources is limited. The main reasons include the NP-hard nature of the problem and the existing resource bottleneck. Therefore, a valid service path cannot always be found even when a valid path exists, and the end-to-end connection requests are blocked from the beginning [5]. To solve the existed resource bottleneck problem, we propose two iterative algorithms in this paper for efficient allocation of additional resources in order to improve the performance in terms of *average request acceptance ratio*, denoted by $\overline{\mathrm{AR}}$. To this end, we (1) introduce a new concept of sensitivity for each service node to locate the bottleneck node, (2) state the problem of allocating additional resources into the service network based on sensitivity, and (3) use sensitivity to propose a simple iterative algorithm and an utilization-based iterative algorithm for efficient allocation of additional resources.

4.1. Definition of Sensitivity. In the service network, a service node can perform complicated data processing functions. In addition, each service node has different resources, for example, CPU capacity, processing power, available services, storage, and memory. The *sensitivity* of a service node represents the impact that this service node has on the performance of entire network (e.g., the impact on average request acceptance ratio). When the most *sensitive* service node (bottleneck node) is located, the owner of the service network (e.g., Infrastructure Provider (InP)) has an opportunity to improve the performance of the entire network by simply supplementing additional resource capacities into one node.

To calculate the sensitivity of a service node, we remove or shut down one different service node n_i each time from the service network and maintain the network working to measure the average request acceptance ratio without n_i, denoted by $\overline{\mathrm{AR}}(i)$. If the $\overline{\mathrm{AR}}(i)$ drops significantly, the service node n_i plays a vital role in the service network and holds high sensitivity.

The set of sensitivities for all service nodes in the service network (G) is a vector defined as $\mathrm{Sen} = (\mathrm{Sen}_0, \mathrm{Sen}_1, \ldots, \mathrm{Sen}_i, \ldots, \mathrm{Sen}_{|N|-1})$ where the element Sen_i representing the sensitivity of service node n_i is defined as

$$\mathrm{Sen}_i = \overline{\mathrm{AR}} - \overline{\mathrm{AR}}(i), \quad \forall n_i \in N. \quad (7)$$

In the resource constrained network, the average request acceptance ratio is a significant performance metric which determines how many end users' requests are accepted.

After the calculation of every service node's sensitivity, we identify the service node with the greatest sensitivity and term it as *the most sensitive node*. We term the adjacent service link of the most sensitive node as *sensitive link*. Then we focus on increasing the CPU capacity of the most sensitive node or the bandwidth capacity of the sensitive links to improve the average request acceptance ratio of the entire network.

4.2. Problem Statement. The problem of allocating additional resources based on sensitivity is stated as follows. We first define the total amount of additional resources added into the service network as

$$\delta(\text{Res}) = \alpha \cdot P(n_\chi) + \beta \cdot B(l_\chi)^T, \tag{8}$$

where the most sensitive node is represented by n_χ and l_χ is a vector representing λ adjacent sensitive links of n_χ defined as $l_\chi = (l_{\chi_1}, l_{\chi_2}, \ldots, l_{\chi_\lambda})$. $B(l_\chi)$ is also a vector representing the bandwidth of each sensitive link defined as $B(l_\chi) = (B(l_{\chi_1}), B(l_{\chi_2}), \ldots, B(l_{\chi_\lambda}))$. α is an integer indicating that the CPU capacity of the most sensitive node will be increased by α times. β is a vector defined as $\beta = (\beta_1, \beta_2, \ldots, \beta_\iota, \ldots, \beta_\lambda)$, where the element β_ι is an integer indicating that the bandwidth capacity of the sensitive link l_{χ_ι} will be increased by β_ι times.

Once the most sensitive node is located, the main objective is to devise the algorithms for efficient allocation of additional resources to improve the performance of entire network.

Similar to the previous work in [15, 19, 20], the revenue (i.e., economic profit) of accepting an end user's request (R) at time t can be defined as the resources that R requires multiplied by their prices

$$\mathbb{R}(R, t) = \sum_{\text{sl}_j \in \text{sl}} p(\text{sl}_j) \cdot \mu_p + b \cdot (\text{sn} + 1) \cdot \mu_b, \tag{9}$$

where μ_p represents the CPU capacity usage price per required resource unit (e.g., \$/instance·hour) and μ_b represents the bandwidth usage price per required resource unit (e.g., \$/Gb·hour). Given that $\mathbb{R}(R, t)$ represents the total price that the end user needs to pay to the InP.

The cost of building a service path for an end user's request at time t can be defined as the total amount of resource capacity that the InP allocates to the service path $\mathscr{P}^{e2e}$ multiplied by their costs

$$\mathbb{C}(R, t) = \sum_{\text{sl}_j \in \text{sl}, n = \mathscr{M}_S(\text{sl}_j)} p(\text{sl}_j) \cdot c(n) + \text{hops}(\mathscr{P}^{e2e}) \cdot b \cdot c(l), \tag{10}$$

where $c(n)$ represents the CPU capacity usage cost per used resource unit (e.g., \$/instance·hour) and $c(l)$ represents the link bandwidth usage cost per used resource unit (e.g., \$/Gb·hour). The cost of serving an end user's request mainly depends on the hops of the chosen service path.

Accordingly, the cost per time unit caused by adding additional resources to the service network is defined as the total amount of additional resources multiplied by theirs costs

$$\mathbb{C}(\delta(\text{Res})) = \alpha \cdot P(n_\chi) \cdot c(n_\chi) + \beta \cdot B(l_\chi)^T \cdot c(l_\chi). \tag{11}$$

After a service path is established, the resources allocated to it will be occupied in the whole lifetime of the corresponding request. Thus the total revenue and cost of serving an end user's request are determined by its lifetime t_d, denoted by $\mathbb{R}(R, t) \cdot t_d$ and $\mathbb{C}(R, t) \cdot t_d$, respectively.

In general, the additional resources are allocated permanently into the service network. Hence, the total cost is determined by the running time $\mathbb{T}$ of the service network, denoted by $\mathbb{C}(\delta(\text{Res})) \cdot \mathbb{T}$.

From InP's point of view, an effective and efficient algorithm of allocating additional resources would minimize the amount of additional resources and maximize the average request acceptance ratio and the average revenue of the InP in the long run. The long-term average revenue of the InP, denoted by $\mathbb{R}(G)$, is defined as

$$\mathbb{R}(G) = \lim_{\mathbb{T} \to \infty} \frac{\sum_{t=0}^{\mathbb{T}} \mathbb{R}(R, t) * t_d}{\mathbb{T}}. \tag{12}$$

The average request acceptance ratio ($\overline{\text{AR}}$) of the service network is defined as

$$\overline{\text{AR}} = \lim_{\mathbb{T} \to \infty} \frac{\sum_{t=0}^{\mathbb{T}} |R_{\text{accepted}}|}{|R|}, \tag{13}$$

where $|R_{\text{accepted}}|$ is the number of requests successfully accepted by the service network and $|R|$ is the total number of requests.

Consider, using the sensitivity-based iterative algorithms for allocating additional resources into the service network, the long-term average cost of the InP which should take the cost caused by taking additional resources into account, denoted by $\mathbb{C}(G)$, is defined as

$$\mathbb{C}(G) = \lim_{\mathbb{T} \to \infty} \frac{\sum_{t=0}^{\mathbb{T}} \mathbb{C}(R, t) * t_d + \mathbb{C}(\delta(\text{Res})) \cdot \mathbb{T}}{\mathbb{T}}. \tag{14}$$

We measure the efficiency of allocating additional resources in terms of the ratio of long-term average revenue to cost ($\mathbb{R}/\mathbb{C}$) ratio which is defined as

$$\frac{\mathbb{R}(G)}{\mathbb{C}(G)} = \lim_{\mathbb{T} \to \infty} \frac{\sum_{t=0}^{\mathbb{T}} \mathbb{R}(R, t) * t_d}{\sum_{t=0}^{\mathbb{T}} \mathbb{C}(R, t) * t_d + \mathbb{C}(\delta(\text{Res})) \cdot \mathbb{T}}. \tag{15}$$

Our objective is to minimize the amount of additional resources ($\delta(\text{Res})$) allocated into service network and accept the largest possible number of end user's requests. We also want to increase the long-term average revenue of InP ($\mathbb{R}(G)$) and decrease the long-term average cost of InP ($\mathbb{C}(G)$). When the average request acceptance ratios of proposed algorithms are nearly the same, we prefer the one that supplements the least amount of resources ($\delta(\text{Res})$) and offers highest long-term $\mathbb{R}/\mathbb{C}$ ratio.

```
(1) Compute and record $\overline{\mathrm{AR}}$, $\mathbb{R}(G)$, $\mathbb{C}(G)$, $\overline{U_N}$, $\overline{U_L}$ using LG-CT($G$)
(2) for all $n_i \in N$ do
(3)     $G_i \leftarrow$ Remove one different $n_i$ each time from $G$
(4)     Compute $\overline{\mathrm{AR}}(i)$ using LG-CT($G_i$)
(5)     $\mathrm{Sen}_i \leftarrow \overline{\mathrm{AR}} - \overline{\mathrm{AR}}(i)$
(6) end for
(7) Locate the most sensitive node $n_\chi$ which is the maximum of Sen
```

Algorithm 1: The sensitivity computing method.

The problem of allocating additional resources stated above is a multiobjective optimization problem with conflicting objectives, which is a combinatorial optimization problem known as NP-hard [21]. As a matter of fact, we can only achieve balance among all above objectives by designing effective and efficient algorithms. For example, we cannot supplement additional resources unlimitedly although the average request acceptance ratio ($\overline{\mathrm{AR}}$) and long-term average revenue ($\mathbb{R}(G)$) increase sharply at the beginning. With the increase of $\delta(\mathrm{Res})$, (1) the corresponding cost ($\mathbb{C}(\delta(\mathrm{Res}))$) which is proportional to $\delta(\mathrm{Res})$ increases; (2) the increase in $\overline{\mathrm{AR}}$ and $\mathbb{R}(G)$ will converge eventually; (3) the increase in $\mathbb{C}(G)$ will significantly exceed the increase in $\mathbb{R}(G)$ from some time. Consequently, the $\mathbb{R}/\mathbb{C}$ will eventually reach an unrealistic value (e.g., $\mathbb{R}(G) < 0.5$) which is unacceptable for an InP. To achieve better performance, we devise two iterative algorithms for allocating additional resources based on the computation of sensitivity, denoted by SI-AAR and UI-AAR, respectively. We will discuss these two algorithms in the following Section 5 in detail.

4.3. Measurement of Resources. To allocate additional resources efficiently, some resource metrics need to be defined and calculated in advance.

4.3.1. Resources on Service Node. The available capacity of a service node, denoted by $A_N(n_i)$, is defined as the available CPU capacity of the service node $n_i \in N$,

$$A_N(n_i) = P(n_i) - \sum_{\forall \mathrm{sl}_j \to n_i \in M_i} p(\mathrm{sl}_j). \quad (16)$$

The capacity utilization ratio of a service node, denoted by $U_N(n_i)$, is defined as the total amount of CPU capacity allocated to different services performed on the service node $n_i \in N$ divided by the CPU capacity of service node $P(n_i)$,

$$U_N(n_i) = \frac{P(n_i) - A_N(n_i)}{P(n_i)}. \quad (17)$$

The average utilization ratio of all service nodes is defined as the summation of utilization ratio of all service node divided by the number of service nodes,

$$\overline{U}_N = \frac{\sum_{i=0}^{|N|-1} U_N(n_i)}{|N|}. \quad (18)$$

4.3.2. Resources on Service Link. Similarly, the available capacity of a service link, denoted by $A_L(l)$, is defined as the total amount of bandwidth available on the service link $l \in L$,

$$A_L(l) = B(l) - b * at. \quad (19)$$

The capacity utilization ratio of a service link, denoted by $U_N(l)$, is defined as the total amount of link bandwidth allocated to different links in $\mathscr{P}^{e2e}$ divided by the bandwidth of the service link $B(l)$,

$$U_L(l) = \frac{B(l) - A_L(l)}{B(l)}. \quad (20)$$

The average utilization ratio of all service links is defined as

$$\overline{U}_L = \frac{\sum_{\forall l \in L} U_L(l)}{|L|}. \quad (21)$$

5. Sensitivity-Based Iterative Algorithms for Allocating Additional Resources

5.1. The Sensitivity Computing Method. The main task of this algorithm (Algorithm 1) is to set up the layered graph and run the capacity tracking algorithm known as *layered graph with capacity tracking* (*LG-CT*) to record the performance metrics of the service network, for example, average request acceptance ratio ($\overline{\mathrm{AR}}$), long-term average revenue ($\mathbb{R}(G)$), long-term average cost ($\mathbb{C}(G)$), average node utilization ($\overline{U}_N$), and average link utilization ($\overline{U}_L$). We then remove one different service node n_i each time from the service network (G) and run LG-CT again to compute corresponding $\mathrm{Sen}_i \in$ Sen. The most sensitive node n_χ is the greatest element in the vector Sen.

Taking advantage of locating the most sensitive node, then we design two iterative algorithms for allocation of additional resources called SI-AAR and UI-AAR, both taking the service network as input. We only consider the supplement of additional resources into the most sensitive node and sensitive links in these two algorithms in order to avoid the rise of the average cost of the InP and the drop of the $\mathbb{R}/\mathbb{C}$ ratio of the InP in the long run. The iteration method is used for additional resources allocation, since the exact values of α and β are impossible to predict directly. On the one hand, inadequate additional resources can not provide significant improvement on performance. On the other hand,

```
(1) Locate the most sensitive node using Algorithm 1
(2) $t_{\mathrm{cpu}} = 0, t_{\mathrm{bw}} = 0, \alpha = 0, \beta' = (1, 1, \dots, 1_{\lambda}), \beta = 0 \cdot \beta'$
(3) repeat
(4)     $t_{\mathrm{cpu}} = t_{\mathrm{cpu}} + \Delta t_{\mathrm{cpu}}, \alpha = t_{\mathrm{cpu}}$
(5)     $t_{\mathrm{bw}} = t_{\mathrm{bw}} + \Delta t_{\mathrm{bw}}, \beta = t_{\mathrm{bw}} \cdot \beta'$
(6)     Add $\delta(\mathrm{Res})$ into $G$
(7)     Call LG-CT($G$)
(8) untile $(\Delta\overline{\mathrm{AR}} < \epsilon)$ or $(\mathbb{R}/\mathbb{C} < \sigma)$
(9) $\alpha = t_{\mathrm{cpu}} - \Delta t_{\mathrm{cpu}}, \beta = (t_{\mathrm{bw}} - \Delta t_{\mathrm{bw}}) \cdot \beta'$
(10) Reset $G$ and supplement recalculated $\delta(\mathrm{Res})$ into $G$
```

Algorithm 2: SI-AAR.

```
(1) Locate the most sensitive node using Algorithm 1
(2) if $\Delta\overline{U} \geq \omega$ then
(3)     Add only CPU capacity into $G$ using Algorithm 4
(4) else if $\Delta\overline{U} \leq -\omega$ then
(5)     Add only Bandwidth capacity into $G$ using Algorithm 5
(6) else
(7)     Add both CPU and bandwidth capacity into $G$ using Algorithm 6
(8) end if
```

Algorithm 3: UI-AAR.

excessive additional resources can result in an overuse of resources. Iteration method provides an effective way to find proper values of α and β by gradually increasing the resource capacity. Details of these two algorithms are given below.

5.2. Simple Iterative Algorithm for Allocating Additional Resources (SI-AAR). SI-AAR (Algorithm 2) describes a simple way to iteratively increase both CPU capacity and bandwidth capacity simultaneously. The algorithm begins by locating the most sensitive node n_χ in service work (G). SI-AAR then supplements $\alpha \cdot P(n_\chi)$ CPU capacity into n_χ and $\beta \cdot B(l_\chi)$ bandwidth capacity into l_χ. Next, it reruns LG-CT and calculates $\mathbb{R}/\mathbb{C}$ ratio and the increment in $\overline{\mathrm{AR}}$ denoted by $\Delta\overline{\mathrm{AR}}$. For each iteration, SI-AAR compares $\Delta\overline{\mathrm{AR}}$ with a small positive fraction ϵ and $\mathbb{R}/\mathbb{C}$ with a threshold parameter σ which is also a positive fraction. The algorithm terminates under two conditions: (1) if $\Delta\overline{\mathrm{AR}} < \epsilon$, $\overline{\mathrm{AR}}$ converges; (2) if $\mathbb{R}/\mathbb{C} < \sigma$, the algorithm will become unacceptable from the point of economic profit. Otherwise, SI-AAR enters the next iteration.

To adjust the increment in additional resources added into the service network in each iteration, we introduce two increment parameters denoted by Δt_{cpu} and Δt_{bw}. Both of them are integers greater than zero and indicate the increment in the amount of CPU capacity and bandwidth capacity, respectively, in each iteration. In realistic network environment, the capacities of network resources (e.g., CPU, storage, bandwidth) can be supplemented by the number of instances which imply the times of resource capacity. For example, we can increase CPU capacity by adding one or multiple CPU instances, increase storage capacity by adding one or multiple hard disks, and increase bandwidth capacity by adding one or multiple communication links. Therefore, it is reasonable to increase resource capacity by one or multiple times in each iteration. β' is a constant vector with all elements having the same value 1 and its size is the same as that of β. $\beta = t_{\mathrm{bw}} \cdot \beta'$ indicates that the bandwidth capacity of all sensitive links will be increased by the same times.

5.3. Utilization-Based Iterative Algorithm for Allocating Additional Resources (UI-AAR). The average utilization ratio of service nodes and service links reflects how much capacity of the network resources is used and which resource is insufficient. The value of the average utilization ratio is decided by several factors, for example, the arrival rate of requests, required resources, and available capacities. We observe the recorded average node utilization ratio $\overline{U}_N$ and average link utilization raio $\overline{U}_L$ and then compute the difference between them, denoted by $\Delta\overline{U}$, defined as $\Delta\overline{U} = \overline{U}_N - \overline{U}_L$. We find that much higher increase in average request acceptance ratio can be achieved efficiently by only adding the resource capacity with higher average utilization ratio while reducing the cost caused by adding additional resources. For example, if $\Delta\overline{U} = 30\%$, the CPU capacity is the scarce resource under much higher stress, and we can improve average request acceptance ratio significantly by only supplementing CPU capacity into the most sensitive node. In this case, average request acceptance ratio increases slightly if we only add bandwidth capacity into sensitive links.

UI-AAR (Algorithm 3) introduces a mechanism to supplement additional resources into the most sensitive node and sensitive links separately or simultaneously as far as the increment and threshold parameters allow. To allocate additional resources into the service network, UI-AAR has three choices:

```
(1) $t_{\text{cpu}} = 0, \alpha = 0, \beta = (0, 0, \ldots, 0)$
(2) repeat
(3)     $t_{\text{cpu}} = t_{\text{cpu}} + \Delta t_{\text{cpu}}, \alpha = t_{\text{cpu}}$
(4)     Add $\delta(\text{Res})$ into $G$
(5)     Call LG-CT($G$)
(6) until $(\Delta\overline{\text{AR}} < \epsilon)$ or $(\mathbb{R}/\mathbb{C} < \sigma)$
(7) $\alpha = t_{\text{cpu}} - \Delta t_{\text{cpu}}$
(8) Reset $G$ and supplement recalculated $\delta(\text{Res})$ into $G$
```

Algorithm 4: AACC.

```
(1) $flag = true, \alpha = 0, \beta' = (1, 1, \ldots, 1_\lambda), \beta = t^v_{\text{bw}} = 0 \cdot \beta'$
(2) while $flag$ do
(3)   for all $l_{\chi_\iota} \in l_\chi$ do
(4)     repeat
(5)       $t^v_{\text{bw}_\iota} = t^v_{\text{bw}_\iota} + \Delta t_{\text{bw}}, \beta'_\iota = t^v_{\text{bw}_\iota}$
(6)       Add $\beta'_\iota \cdot B(l_{\chi_\iota})$ bandwidth capacity into $G$
(7)       Call LG-CT($G$)
(8)     until $(\Delta\overline{\text{AR}} < \epsilon_b)$ or $(\mathbb{R}/\mathbb{C} < \sigma)$
(9)     $\beta'_\iota = t^v_{\text{bw}_\iota} = t^v_{\text{bw}_\iota} - \Delta t_{\text{bw}}$
(10)    Record $\overline{\text{AR}}(l_{\chi_\iota})$ which represents the $\overline{\text{AR}}$ after adding $\beta'_\iota \cdot B(l_{\chi_\iota})$ bandwidth capacity into $G$
(11)    Reset $G$
(12)  end for
(13)  if all elements in $t^v_{\text{bw}}$ do not change then
(14)    $flag = false$
(15)  else
(16)    Locate the $l_{\chi_k}$ which has the greatest $\overline{\text{AR}}(l_{\chi_k})$
(17)    $\beta_k = \beta'_k, t^v_{\text{bw}} = \beta, \beta' = (1, 1, \ldots, 1_\lambda)$
(18)    Supplement recalculated $\delta(\text{Res})$ into $G$
(19)  end if
(20) end while
```

Algorithm 5: AABC.

(1) adding CPU capacity to the most sensitive node;

(2) adding bandwidth capacity to sensitive links;

(3) adding both CPU capacity and bandwidth capacity.

UI-AAR makes a decision according to the value of $\Delta\overline{U}$ and the parameter ω which is a positive fraction and represents the threshold of $\Delta\overline{U}$. We compare $\Delta\overline{U}$ with ω to determine which resource capacity should be added. We will discuss the three scenarios as follows:

(1) If $\Delta\overline{U} \geq \omega$, UI-AAR only supplements the CPU capacity into the most sensitive node using Algorithm 4: *Allocating Additional CPU Capacity* (*AACC*).

The process is the same as that in SI-AAR if $\Delta t_{\text{bw}} = 0$.

(2) If $\Delta\overline{U} \leq -\omega$, UI-AAR only supplements the bandwidth capacity into sensitive links using Algorithm 5: *Allocating Additional Bandwidth Capacity* (*AABC*).

Generally, the most sensitive node has more than one sensitive links. We cannot increase the bandwidth capacities of all sensitive links simultaneously by the same times t_{bw} since it is not cost-efficient (i.e., adding bandwidth capacity to some sensitive link makes no contribution to the performance in terms of acceptance ratio). AABC only selects one sensitive link per iteration and adds bandwidth capacity into it. Like Algorithm 4, for each sensitive link, AABC gradually supplements its bandwidth capacity until $\overline{\text{AR}}$ converges. Then AABC records $\overline{\text{AR}}(l_{\chi_\iota})$ which represents the average request acceptance ratio achieved by adding $t^v_{\text{bw}_\iota}$ times of bandwidth capacity to the corresponding sensitive link. After dealing with the last sensitive link, AABC identifies the sensitive link which has the greatest $\overline{\text{AR}}(l_{\chi_\iota})$, increases its bandwidth capacity by $t^v_{\text{bw}_\iota}$ times, and writes it back to the service network topology. This process will be repeated until all $\overline{\text{AR}}(l_{\chi_\iota})$ converge. That is to say, adding additional bandwidth capacity to any sensitive link can not increase $\Delta\overline{\text{AR}}$ over ϵ_b. t^v_{bw} is a vector with the same size as that of β. Each element $t^v_{\text{bw}_\iota} \in t^v_{\text{bw}}$, of which value represents the times of bandwidth capacity added into sensitive links l_{χ_ι}, can have different value.

(3) If $-\omega < \Delta\overline{U} < \omega$, UI-AAR supplements the CPU capacity into the most sensitive node and the bandwidth capacity into sensitive links simultaneously using Algorithm 6: *Allocating Additional CPU and Bandwidth Capacity* (*AACBC*).

```
(1) $t_{cpu} = 0, \alpha = 0, \beta = (0, 0, \ldots, 0_\lambda)$
(2) **repeat**
(3)     $t_{cpu} = t_{cpu} + \Delta t_{cpu}, \alpha = t_{cpu}$
(4)     Add $\alpha \cdot P(n_\chi)$ CPU capacity into $G$
(5)     Computing $\beta$ using Algorithm 5 where, in the first line,
        (1) $\beta$ do not be reset to $(0, 0, \ldots, 0_\lambda)$; (2) set $t^v_{bw} = \beta$.
(6)     Supplement $\delta(\text{Res})$ into $G$
(7)     Call LG-CT($G$)
(8) **until** $(\Delta\overline{\text{AR}} < \epsilon)$ or $(\mathbb{R}/\mathbb{C} < \sigma)$
(9) $\alpha = t_{cpu} - \Delta t_{cpu}$
(10) Reset $G$ and supplement recalculated $\delta(\text{Res})$ into $G$
```

Algorithm 6: AACBC.

Table 2: Differences in three settings.

	Setting I	Setting II	Setting III
Required CPU capacity	(0, 50][a]	(0, 20][a]	(0, 50][a]
Required bandwidth	(0, 20][a]	(0, 50][a]	(0, 50][a]

[a]The values are real numbers uniformly distributed on the corresponding range.

AACBC combines the method in Algorithm 4 with the method in Algorithm 5. Like the method in Algorithm 4, for the most sensitive node, AACBC gradually increases its CPU capacity by α times. For each value of α, AACBC uses the method in Algorithm 5 to compute β and then supplements the corresponding resources $\delta(\text{Res})$ into service network. If the condition is true, AACBC adds the value of Δt_{cpu} to α and repeats the above process until $\overline{\text{AR}}$ converges.

6. Performance Evaluation

In this section, we first describe the evaluation environment and then present our main experimental results to evaluate efficiency of the sensitivity-based iterative algorithms in terms of average request acceptance ratio, allocated additional resources, long-term average revenue, long-term average cost, and $\mathbb{R}/\mathbb{C}$ ratio. Our evaluation primarily focuses on the performance comparison between proposed two algorithms and the advantages of sensitivity-based bottleneck locating.

6.1. Evaluation Environment. We have implemented a discrete event simulator to evaluate our proposed algorithms. Three different settings are chosen for the following experiments. Differences among the three settings are introduced in Section 6.1.3 and shown in Table 2.

6.1.1. Service Network Topology. In this paper, we do not assume any specialized network topologies. We first use the GT-ITM Tool [22] to randomly generate service network topologies. Each service network is a 20-node 27-link topology with a choice of 4 different services, a scale that corresponds to a small-sized ISP. Specifically, the number of services available on one single service node is an integer uniformly distributed between 1 and 4. The bandwidth capacity of service links and the CPU capacity of service nodes are real numbers uniformly distributed between 50 and 100. The communication delay of each service link is an integer which is proportional to its Euclidean distance and normalized between 1 and 10. Each service's processing time is an integer which depends on its type and the CPU power of its corresponding service node and normalized between 1 and 10.

6.1.2. Request. We assume that end user's requests arrive in a Poisson process with an average rate of 4 requests per 100 time units. Each end user's request has an exponentially distributed duration time with an average of 1000 time units. We run our simulation for about 50,000 time units, which corresponds to about 2,000 requests, for an instance of the simulation. The required bandwidth for service links and the required CPU capacity for each service are configured according to different settings shown in Table 2. The number of services in end user's requests is fixed to 4. The ordered services list consists of 4 different services randomly distributed among S_1, S_2, S_3, and S_4.

6.1.3. Differences in Three Settings. To observe the performance of our algorithms under different resource utilization, three different experimental settings are configured for our simulation. The differences among them only exist in required CPU capacity for each service and required bandwidth for service links. Setting I represents the scenario that the service network is under more pressure from requested CPU capacity resources than it has from requested bandwidth resources. On the contrary, resource stress that the service network encounters mainly stems from the requested bandwidth in Setting II. In Setting III, with the increasing requirements for resources, the available capacities on both service nodes and service links become insufficient to accept more requests. Details are shown in Table 2.

6.2. Compared Algorithms and Parameter Settings. In our simulator, we implement our sensitivity-based iteration algorithms for allocating additional resources alongside the related strategies: (1) SI-AAR and (2) UI-AAR. We use two specific cases of SI-AAR, denoted by SI-AAR-Least and SI-AAR-RUB (referenced upper bound, RUB), to provide a lowest bound and an referenced upper bound, respectively, on the amount of additional resources. In SI-AAR-Least, α is set to 1 and β is set to $(1, 1, \ldots, 1)$, which represents the situation of adding the least amount of resources to the service network using SI-AAR. In SI-AAR-RUB, α is set to 9 and β is set to $(9, 9, \ldots, 9)$; that is, the amount of resources of the most sensitive node and sensitive links is increased to 10 times which is not practical but provides a referenced upper bound on the performance of realistic algorithms. Based on extensive simulations, we adjust the parameters of proposed two algorithms. We set μ_p, μ_b, $c(n)$, and $c(l)$ to 1, Δt_{cpu} and Δt_{bw} to 1, ϵ to 1%, ϵ_b to 0.5%, σ to 0.6, and ω to 10% to achieve greatest increase in average request acceptance ratio and to decrease the amount of additional resources and the number of iterations in the meantime.

6.3. Performance Metrics. In our experiments, we use several performance metrics introduced in previous sections for the purpose of evaluation, for example, average request acceptance ratio ($\overline{\text{AR}}$), the amount of allocated additional resources ($\delta(\text{Res})$), long-term average revenue ($\mathbb{R}(G)$), long-term average cost ($\mathbb{C}(G)$), and long-term $\mathbb{R}/\mathbb{C}$ ratio. We measure the average request acceptance ratio ($\overline{\text{AR}}$) and average request acceptance ratio without node i ($\overline{\text{AR}}(i)$), to compute the sensitivities and locate the most sensitive node in the original service network (i.e., the service network without any additional resources being added). To evaluate the effectiveness and the efficiency of our proposed algorithms, we also measure the average request acceptance ratio, average revenue, average cost, and $\mathbb{R}/\mathbb{C}$ ratio for end user's requests over time in the service network where the corresponding additional resources have been added to. In the meantime, we record the values of two essential parameters α and β to compute the amount of additional resources ($\delta(\text{Res})$) eventually allocated into the service network for evaluated algorithms.

6.4. Evaluation Results. We present the evaluation results to show the effectiveness and quantify the efficiency of the proposed algorithms under three different scenarios (depicted in Figures 2, 3, and 4).

We first plot $\overline{\text{AR}}$ and $\overline{\text{AR}}(i)$ against the index of service nodes to show the computation of sensitivities (depicted in Figures 2(a), 3(a), and 4(a)). We use a bar chart to compare the amount of allocated additional resources ($\delta(\text{Res})$) (depicted in Figures 2(b), 3(b), and 4(b)). Next, we plot average request acceptance ratio, average revenue, average cost, and $\mathbb{R}/\mathbb{C}$ ratio against time to show how each of these algorithms actually performs in the long run (depicted in Figures 2(c)–2(f), 3(c)–3(f) and 4(c)–4(f)). We summarize our key observations for the three settings (Table 2) as follows.

6.4.1. Sensitivity Computing. We locate the most sensitive node through computation of sensitivities and choose the strategy of allocating additional resources for UI-AAR according to the results of resource utilization computing (shown in Table 3).

(1) Setting I: as shown in Figure 2(a) and Table 3, node 7 is the most sensitive node, $\overline{U}_N = 34.9\%$, $\overline{U}_L = 23.5\%$, $\Delta\overline{U} = 11.4\% > \omega$, and the CPU capacity of node 7 is chosen to be supplemented in UI-AAR.

(2) Setting II: from Figure 3(a) and Table 3, node 10 is the most sensitive node, $\overline{U}_N = 10.4\%$, $\overline{U}_L = 32.5\%$, $\Delta\overline{U} = -22.1\% < -\omega$, and the bandwidth of node 10's sensitive links is chosen to be supplemented in UI-AAR.

(3) Setting III: the configurations of required CPU capacity and bandwidth (Table 2) show that both CPU capacity and bandwidth capacity are under huge pressure. That is the reason why the average request acceptance ratio is only 64% and $\overline{U}_N$ is close to $\overline{U}_L$ (shown in Figure 4(a) and Table 3). Given that, node 7 is the most sensitive node, $-\omega < \Delta\overline{U} = 3\% < \omega$, and the CPU capacity of node 7 and the bandwidth of node 7's sensitive links are chosen to be supplemented together in UI-AAR.

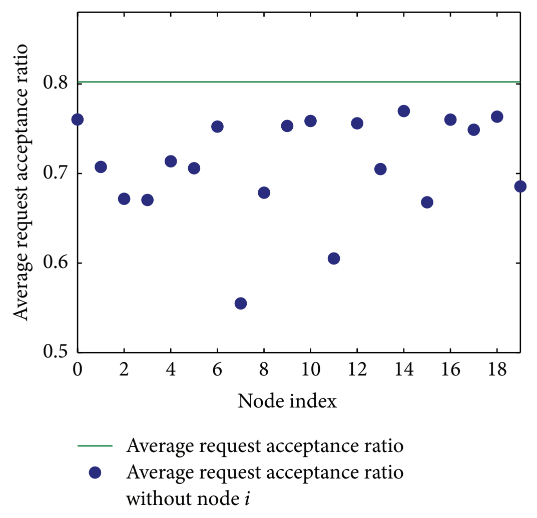
Table 3: Resource utilization in three settings.

	Setting I	Setting II	Setting III
Average node utilization ratio	34.9%	10.4%	33.7%
Average link utilization ratio	23.5%	32.5%	30.8%

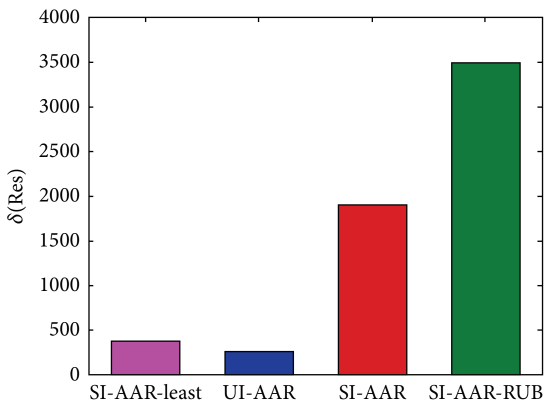
6.4.2. Allocating Additional Resources into the Most Sensitive Nodes and Sensitive Links Leads to Higher Average Request Acceptance Ratio. In Figures 2(a), 3(a), and 4(a), we also plot average request acceptance ratio against time to show how the original LG-CT algorithm, denoted by *Original*, performs without any additional resources being added to the service network. It is important to note that we only present the effectiveness of the proposed algorithms here. We will discuss the efficiency of the proposed algorithms to see if and how they can make efficient use of allocated additional resources in Section 6.4.3.

From Figures 2(a), 3(a), and 4(a), it is evident that the proposed algorithms, UI-AAR and SI-AAR, and the two specific cases, SI-AAR-Least and SI-AAR-RUB, lead to higher average request acceptance ratio than the *Original* under three different scenarios. These three graphs show that sensitivity-based bottleneck locating plays a vital role in allocation of additional resources. It is an effective way to increase the average request acceptance ratio only by supplementing additional resources into bottleneck node (the most sensitive node) and links (sensitive links).

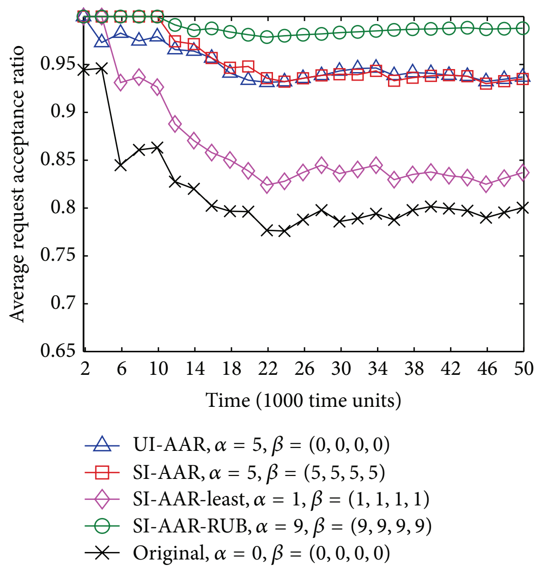
In the three outlined settings, after time unit of 20,000, the average request acceptance ratio of SI-AAR and UI-AAR is nearly the same. In addition, the approximate increments in average request acceptance ratio (e.g., 13.4% and 13.6% in Setting I, 15.6% and 15.4% in Setting II, and 16% and 15.8% in Setting III at the time unit of 50,000) show that both SI-AAR and UI-AAR perform well. In Setting I, SI-AAR-RUB gains much higher acceptance ratio than SI-AAR and UI-AAR, since it supplements ten times amount of additional resources. However, in Setting II and Setting III, the average request acceptance ratio of SI-AAR-RUB is only 0.4% and 2.2% higher than that of SI-AAR and 0.6% and 2.6% higher



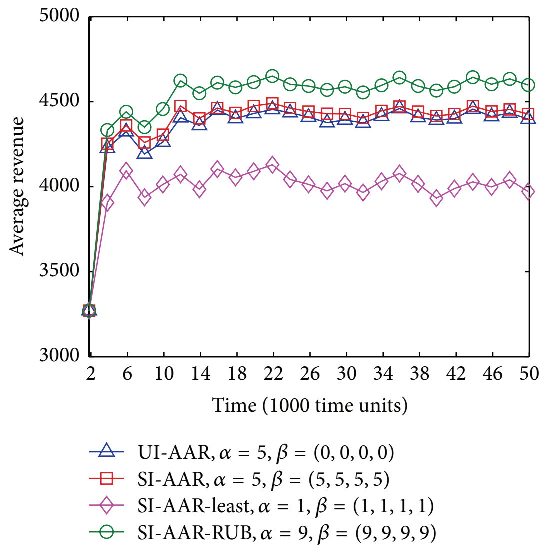
(a) Sensitivity computing



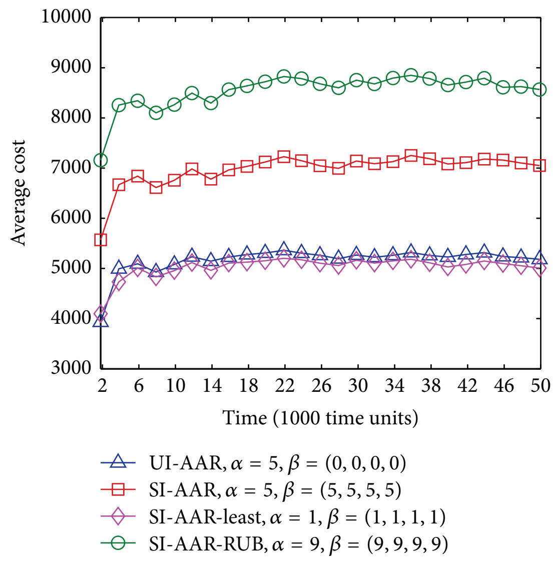
(b) Additional resources allocated into the service network



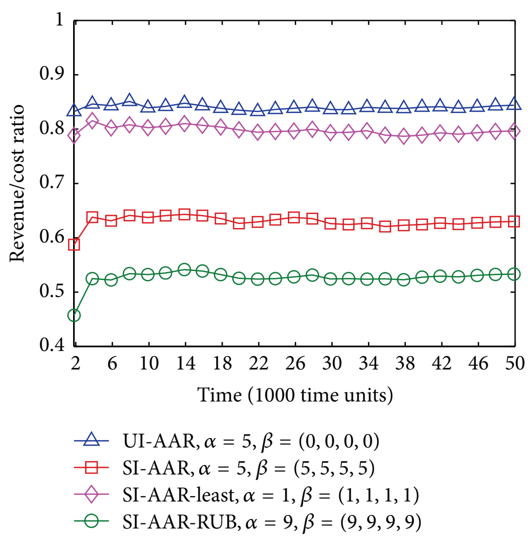
(c) Comparisons of average request acceptance ratios over time



(d) Comparisons of long-term average revenues

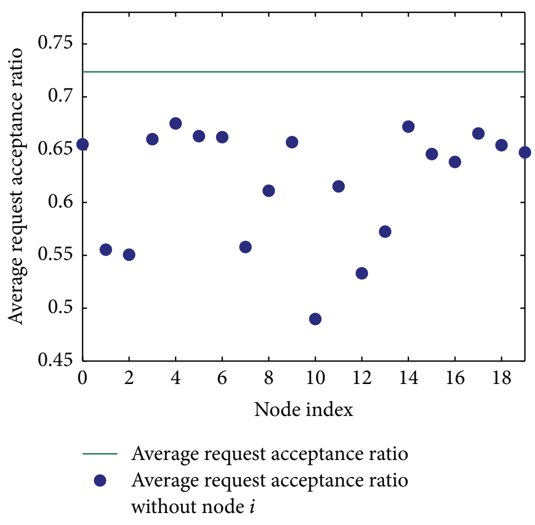


(e) Comparisons of long-term average costs

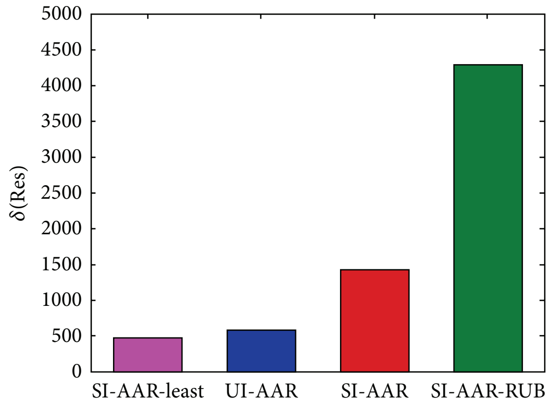


(f) Comparisons of long-term $\mathbb{R}/\mathbb{C}$ ratios

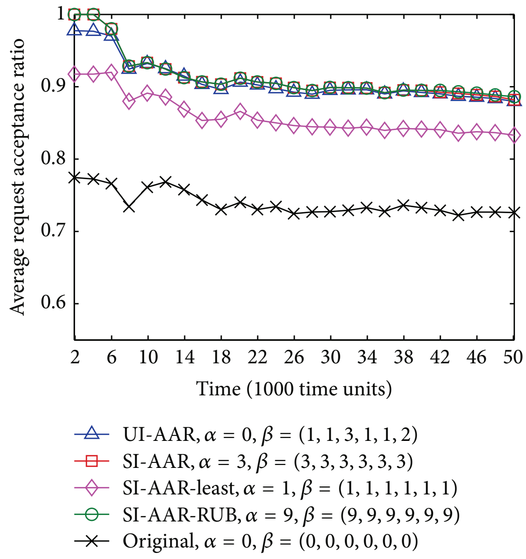
Figure 2: Comparisons between UI-AAR and SI-AAR in Setting I.



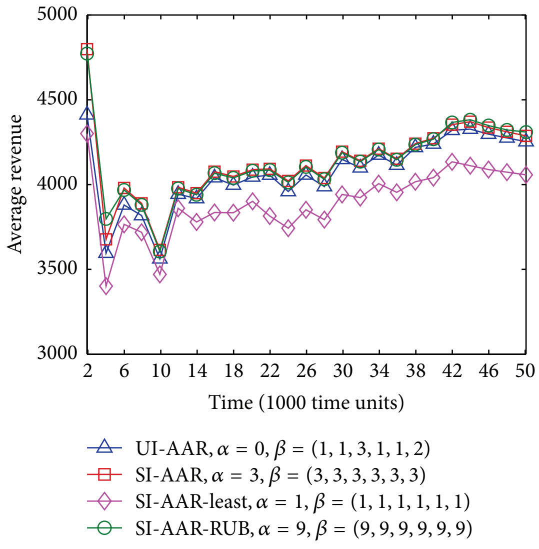
(a) Sensitivity computing



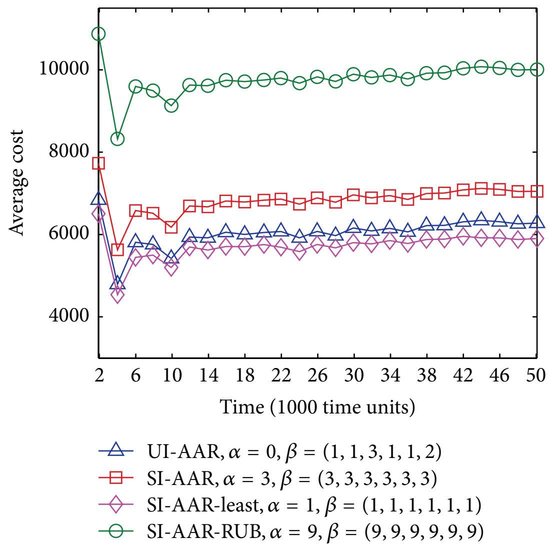
(b) Additional resources allocated into the service network



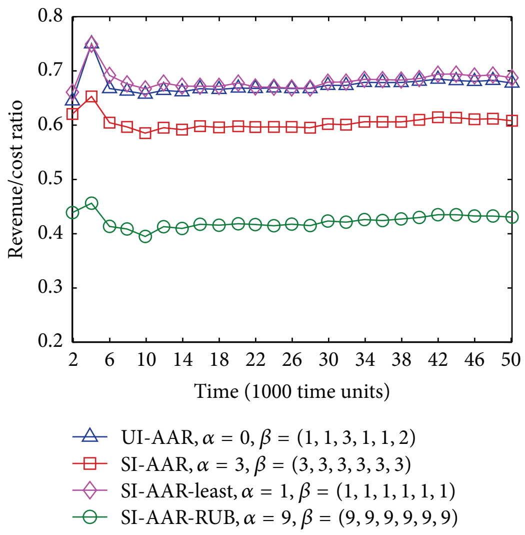
(c) Comparisons of average request acceptance ratios over time



(d) Comparisons of long-term average revenues

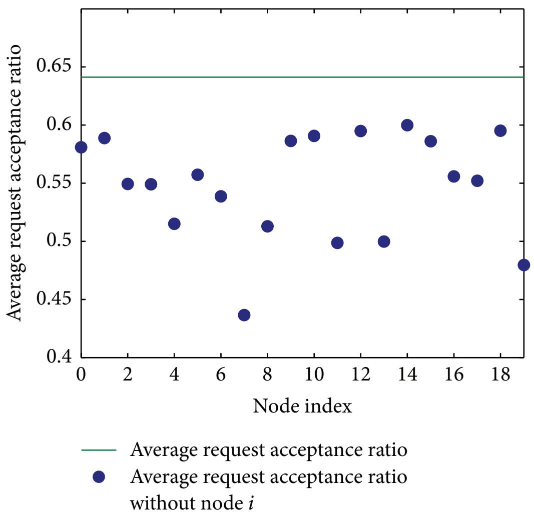


(e) Comparisons of long-term average costs

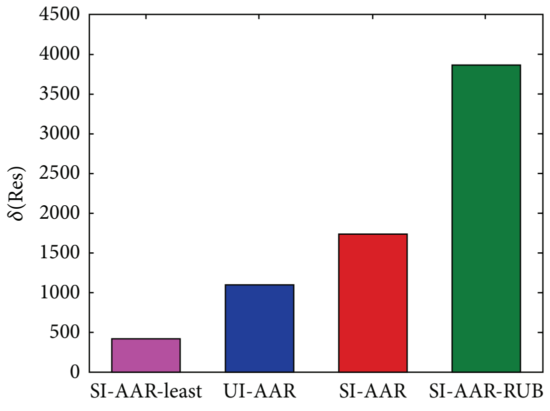


(f) Comparisons of long-term $\mathbb{R}/\mathbb{C}$ ratios

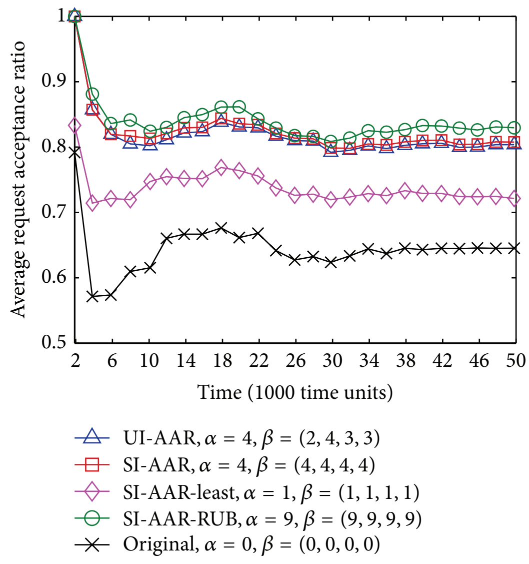
FIGURE 3: Comparisons between UI-AAR and SI-AAR in Setting II.



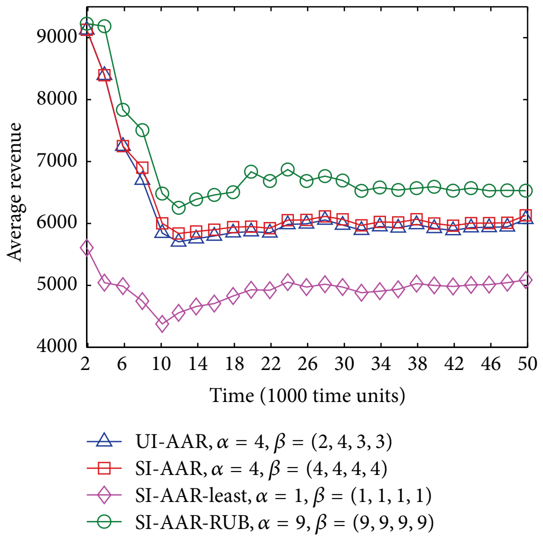
(a) Sensitivity computing



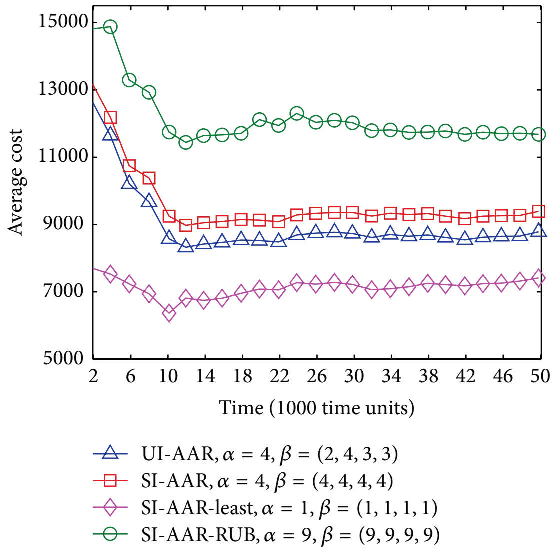
(b) Additional resources allocated into the service network



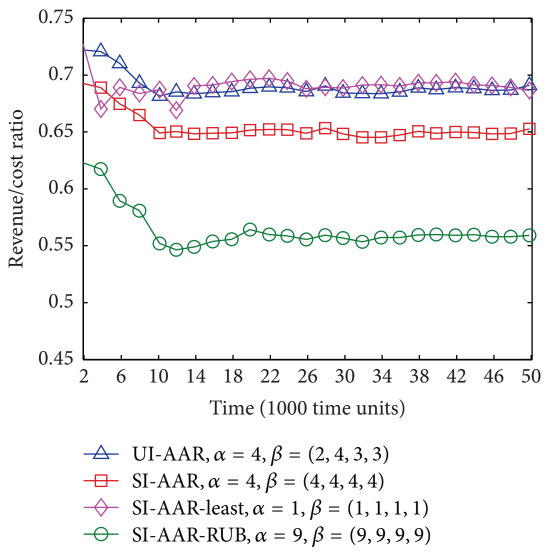
(c) Comparisons of average request acceptance ratios over time



(d) Comparisons of long-term average revenues



(e) Comparisons of long-term average costs



(f) Comparisons of long-term $\mathbb{R}/\mathbb{C}$ ratios

FIGURE 4: Comparisons between UI-AAR and SI-AAR in Setting III.

than that of UI-AAR, respectively. On the contrary, SI-AAR-Least produces the least increase in acceptance ratio among the four algorithms. The reasons with respect to above two situations will be analyzed in Section 6.4.3.

6.4.3. UI-AAR Can Allocate the Additional Resources More Efficiently. It is worth noting that the evaluated algorithms that lead to the higher acceptance ratio also produce higher long-term average revenue (depicted in Figures 2(c), 2(d), 3(c), 3(d), 4(c), and 4(d)) and higher long-term average cost (excluding the cost produced by additional resources). (1) According to the definition of $\mathbb{R}(G)$, more requests are accepted, while more revenue can be obtained, and (2) all of them use the same LG-CT algorithm for solving the service placement problem (i.e., same approaches of service placement and resource allocation). However, the amount of additional resources producing additional cost allocated by the compared algorithms is different, which implies that $\delta(\text{Res})$ and $\mathbb{R}/\mathbb{C}$ ratio are two vital metrics to quantify the efficiency of the evaluated algorithms.

From Figures 2(b), 3(b), and 4(b), as the lowest bound and the referenced upper bound, SI-AAR-Least and SI-AAR-RUB use the least and most amount of additional resources in SI-AAR, respectively. The additional resources used by UI-AAR are close to that used by SI-AAR-Least in Setting I and Setting II and are twice greater in Setting III. SI-AAR allocates more additional resources compared with UI-AAR, for example, 4, 3, and 1.5 times greater in Setting I, Setting II, and Setting III, respectively. The reason is that UI-AAR only adds CPU capacity or bandwidth capacity in Setting I and Setting II, and both SI-AAR and UI-AAR add CPU capacity and bandwidth capacity together in Setting III.

Given that, the evaluated algorithms that lead to the higher acceptance ratio also produce higher additional cost ($\mathbb{C}(\delta(\text{Res}))$ and thus produce higher long-term average cost ($\mathbb{C}(G)$) (depicted in Figures 2(d), 3(d), and 4(d)).

Based on the above illustration and analysis, UI-AAR performs considerably well in terms of acceptance ratio, average revenue, and cost and uses less additional resources. Consequently, UI-AAR produces the highest $\mathbb{R}/\mathbb{C}$ ratio among UI-AAR, SI-AAR, and SI-AAR-RUB (depicted in Figures 2(f), 3(f), and 4(f)). The reasons why UI-AAR can make more efficient use of additional resources are outlined as follow. (1) UI-AAR selectively supplements the additional resources based on resource utilization ratio. For example, in Setting I, UI-AAR select only CPU capacity to supplement in each iteration (see Figure 2(f), where $\alpha = 5$ and $\beta = (0, 0, 0, 0)$), the total amount additional resources is 4 times lower than that of SI-AAR and even less than that of SI-AAR-Least. (2) UI-AAR can find better combination of α and β. For each value of α, UI-AAR computes the optimal value of each element in β. For example, in Setting III, for each value of α, UI-AAR iteratively supplements only one sensitive link's bandwidth capacity in each iteration, and thus the bandwidth capacities eventually added to sensitive links are different despite the value of α being the same as that in SI-AAR (see Figure 3(f), where $\alpha = 4$ and $\beta = (2, 4, 3, 3)$). The average request acceptance ratio of UI-AAR and SI-AAR is nearly the same, but the total amount of additional resources of UI-AAR is 1.5 times lower than that of SI-AAR. Adding more additional resources along with no improvement on acceptance ratio implies that part of additional resources added by SI-AAR does not make any contribution to the performance in terms of acceptance ratio. Likewise, SI-AAR-RUB is a suitable case to illustrate the overuse of resources. Although the $\mathbb{R}/\mathbb{C}$ ratio and $\delta(\text{Res})$ of SI-AAR-Least are close to those of UI-AAR (depicted in Figures 2(b), 2(f), 3(b), 3(f), 4(b), and 4(f)), UI-AAR significantly outperform SI-AAR-Least in terms of the average request acceptance ratio and the long-term average revenue. The reasons are given in the following. (1) The additional resources added by SI-AAR-Least are insufficient to accept more requests. (2) Like SI-AAR, SI-AAR-Least does not allocate the additional resource efficiently. For example, compared with UI-AAR, SI-AAR-Least accepts less requests using more additional resources in Setting I.

7. Conclusion and Future Work

The placement of services is an important problem in any network architecture that supports the implementation of data processing functions inside the network. In this paper, we modeled and stated this problem. To solve the resource bottleneck problem existing in LG-CT algorithm, we introduced a novel concept: *sensitivity*. We used the sensitivity to locate the most sensitive node and then allocated additional resources into the most sensitive node and sensitive links to improve the performance of entire network. After discussing and formulating the problem of allocating additional resources, we proposed two sensitivity-based iterative algorithms for efficient resource allocation. The first one, SI-AAR, provides a simple way to increase both the CPU capacity and the bandwidth capacity by the same times. The second one, UI-AAR, can supplement additional resources selectively based on resource utilization ratio. Our results from the experiments showed the effectiveness and efficiency of our proposed two algorithms under three specific settings. The increase in average request acceptance ratio was significant if we supplemented additional resources to the most sensitive node and sensitive links. The utilization-based iterative algorithm (UI-AAR) can make more efficient use of additional resources and thus outperforms SI-AAR in terms of the amount of allocated additional resources, long-term average cost, and long-term $\mathbb{R}/\mathbb{C}$ ratio.

In future work, we will consider the number of the bottleneck nodes which we choose to supplement resources, focus on medium-size or large-scale network topology, and investigate where the sensitivity can be further applied in the service placement problem to improve the performance in terms of optimizing capacity allocation and balancing load.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research was partially supported by the National Natural Science Foundation of China under Grant no. 61379079 and the National Basic Research Program of China (973) under Grant no. 2012CB315900.

References

[1] A. Feldmann, “Internet clean-slate design: what and why?” *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 3, pp. 59–64, 2007.

[2] T. Wolf, “In-network services for customization in next-generation networks,” *IEEE Network*, vol. 24, no. 4, pp. 6–12, 2010.

[3] S. Ganapathy and T. Wolf, “Design of a network service architecture,” in *Proceedings of the 16th IEEE International Conference on Computer Communications and Networks (ICCCN ’07)*, pp. 754–759, August 2007.

[4] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, “The silo architecture for services integration, control, and optimization for the future internet,” in *Proceedings of the IEEE International Conference on Communications (ICC ’07)*, pp. 1899–1904, June 2007.

[5] S. Choi, J. Turner, and T. Wolf, “Configuring sessions in programmable networks,” *Computer Networks*, vol. 41, no. 2, pp. 269–284, 2003.

[6] M. Vellala, A. Wang, G. N. Rouskas, R. Dutta, I. Baldine, and D. Stevenson, “A composition algorithm for the silo cross-layer optimization service architecture,” in *Proceedings of the 1st International Conference on Advanced Network and Telecommunications Systems (ANTS ’07)*, 2007.

[7] H. H. L. Ruf, K. Farkas, and B. Plattner, “Network services on service extensible routers,” in *Active and Programmable Networks: IFIP TC6 7th International Working Conference, IWAN 2005, Sophia Antipolis, France, November 21-23, 2005. Revised Papers*, Lecture Notes in Computer Science, pp. 53–64, Springer, Berlin, Germany, 2005.

[8] T. Wolf, “Challenges and applications for network-processor-based programmable routers,” in *Proceedings of the IEEE Sarnoff Symposium*, pp. 1–4, March 2006.

[9] T. Anderson, L. Peterson, S. Shenker, and J. Turner, “Overcoming the internet impasse through virtualization,” *Computer*, vol. 38, no. 4, pp. 34–41, 2005.

[10] X. Huang, S. Ganapathy, and T. Wolf, “A distributed routing algorithm for networks with data-path services,” in *Proceedings of 17th IEEE International Conference on Computer Communications and Networks (ICCCN ’08)*, pp. 1–7, 2008.

[11] H. Xin, S. Ganapathy, and T. Wolf, “A scalable distributed routing protocol for networks with data-path services,” in *Proceedings of the 16th IEEE International Conference on Network Protocols (ICNP ’08)*, pp. 318–327, October 2008.

[12] B. Raman and R. H. Katz, “Load balancing and stability issues in algorithms for service composition,” in *Proceedings of the 22nd Annual Joint Conference on the IEEE Computer and Communications Societies (INFOCOM ’03)*, vol. 2, pp. 1477–1487, IEEE, San Francisco, Calif, USA, April 2003.

[13] J. Liang and K. Nahrstedt, “Service composition for advanced multimedia applications,” in *Multimedia Computing and Networking*, vol. 5680 of *Proceedings of SPIE*, pp. 228–240, International Society for Optics and Photonics, San Jose, Calif, USA, January 2005.

[14] Z. Qian, S. Zhang, K. Yim, and S. Lu, “Service oriented multimedia delivery system in pervasive environments,” *Journal of Universal Computer Science*, vol. 17, no. 6, pp. 961–980, 2011.

[15] K.-T. Tran, N. Agoulmine, and Y. Iraqi, “Cost-effective complex service mapping in cloud infrastructures,” in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS ’12)*, pp. 1–8, IEEE, April 2012.

[16] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang, “Locating internet bottlenecks: algorithms, measurements, and implications,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 41–54, 2004.

[17] N. F. Butt, M. Chowdhury, and R. Boutaba, “Topology-awareness and reoptimization mechanism for virtual network embedding,” in *NETWORKING 2010*, vol. 6091 of *Lecture Notes in Computer Science*, pp. 27–39, Springer, Berlin, Germany, 2010.

[18] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, “Vnr algorithm: a greedy approach for virtual networks reconfigurations,” in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM ’11)*, pp. 1–6, IEEE, 2011.

[19] M. Yu, Y. Yi, J. Rexford, and M. Chiang, “Rethinking virtual network embedding: substrate support for path splitting and migration,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[20] M. Shen, K. Xu, K. Yang, and H.-H. Chen, “Towards efficient virtual network embedding across multiple network domains,” in *Proceedings of the 22nd IEEE International Symposium of Quality of Service (IWQoS ’14)*, pp. 61–70, IEEE, May 2014.

[21] M. Ehrgott and X. Gandibleux, “A survey and annotated bibliography of multiobjective combinatorial optimization,” *OR-Spektrum*, vol. 22, no. 4, pp. 425–460, 2000.

[22] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, “How to model an internetwork,” in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM ’06), 15th Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation*, pp. 594–602, March 1996.