Research Article

Analysing the Impact of Scaling Out SaaS Software on Response Time

Bo Dong , **Shi Ying** , **Bingming Wang** , **Guoli Cheng**, and **Jiangyi Geng**

School of Computer Science, Wuhan University, Wuhan 430072, China

Correspondence should be addressed to Shi Ying; yingshi@whu.edu.cn

Received 12 December 2020; Revised 4 March 2021; Accepted 22 March 2021; Published 5 April 2021

Academic Editor: Roberto Natella

Copyright © 2021 Bo Dong et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

When SaaS software suffers from the problem of response time degradation, scaling deployment resources that support the operation of software can improve the response time, but that also means an increase in the costs due to additional deployment resources. For the purpose of saving costs of deployment resources while improving response time, scaling out the SaaS software is an alternative approach. However, how scaling out software affects response time in the case of saving deployment resources is an important issue for effectively improving response time. Therefore, in this paper, we propose a method analysing the impact of scaling out software on response time. Specifically, we define the scaling-out operation of SaaS software and then leverage queueing theory to analyse the impact of the scaling-out operation on response time. According to the conclusions of impact analysis, we further derive an algorithm for improving response time based on scaling out software without using additional deployment resources. Finally, the effectiveness of the analysis's conclusions and the proposed algorithm is validated by a practical case, which indicates that the conclusions of impact analysis obtained from this paper can play a guiding role in scaling out software and improving response time effectively while saving deployment resources.

1. Introduction

1.1. The Motivation and Research Challenges. Response time degradation is one of the software performance issues that SaaS (Software-as-a-Service) providers may encounter due to changes in the runtime environment of the software. As a new software delivery model, SaaS software is composed of a series of atomic services, which are usually deployed on distributed deployment resources and together perform the function of the software [1, 2]. When response time degradation occurs, for the sake of ensuring the efficient operation of the software, SaaS providers usually solve this problem by adopting scaling techniques, for instance, scaling up the deployment resources (i.e., upgrading the configurations of current deployment resources where the software is deployed) or scaling out the deployment resources (i.e., providing more deployment resources for the software). However, that will also result in more costs due to renting additional deployment resources.

For the purpose of saving costs of deployment resources, scaling out SaaS software (i.e., adding more instances of services in software) without using additional deployment resources is another approach to improve the response time. Nevertheless, that may not be an easy task for SaaS providers. In such a situation, the new service instances generated by scaling out the software may have to be deployed on existing deployment resources, and then, resource contention will occur between these service instances, which may have a negative impact on response time. If SaaS providers do not know the appropriate strategies for scaling out software with existing deployment resources, the response time of the software may not be effectively improved, which may also increase the burden of human operation. Therefore, how to scale out SaaS software to effectively improve the response time without increasing the cost of additional deployment resources becomes a challenging issue for SaaS providers.

1.2. The Proposed Solution. To address the issue mentioned above, our solution is to study the impact of scaling out SaaS software on response time without using additional deployment resources, in order to provide guidance for SaaS providers to scale out software and improve response time effectively. For this purpose, a method for performance analysis is necessary. Queueing theory has been widely applied in performance analysis [3, 4]. Since queueing theory is a mathematical analysis method, the performance metrics of the software (for instance, response time) can be analysed in combination with deployment resources and solved with mathematical formulas quantitatively and efficiently, which can contribute to subsequently analysing the impact of scaling out software on response time and making decisions for scaling-out operations [5]. Therefore, in this paper, we use queueing theory for our study. In brief, the key contributions of the paper are summarized as follows:

- (i) We define the scaling-out operation of SaaS software and then analyse the positive and negative influences of scaling-out operations on response time of the software, which gives the direction of scaling out SaaS software
- (ii) According to the analysis conclusions, we put forward an algorithm of scaling out software without using additional deployment resources, which is helpful in effectively improving response time of the software while saving deployment resources
- (iii) Aiming to verify the effectiveness of the analysis conclusions and the proposed algorithm, we set up experiments on a practical case

The rest of this paper is organized as follows. Section 2 describes the background and problem statement of this paper. Section 3 introduces the process of analysing the impact of scaling out SaaS software on response time. Section 4 presents a practical case verifying the effectiveness of the analysis conclusions obtained from our study. The discussion of our study is presented in Section 5. Section 6 summarizes the related work, and our work is concluded in Section 7.

2. Background and Problem Statement

To facilitate the description of scaling out SaaS software and the analysis of the response time of software in combination with deployment resources, in this section, first, we will introduce the concepts of SaaS software, deployment scheme, and deployment resources mentioned in this paper. Based on these concepts, we will give a description of the problem to be addressed in this paper.

2.1. SaaS Software. The SaaS software consists of different types of atomic services based on specific logical interaction relations. Figure 1 shows an example of SaaS software, which is composed of five types of atomic services ($S_1 \sim S_5$). For ease of explanation, we use a tree structure to describe the interaction relationship between atomic services (shown in Figure 2) [6]. In Figure 2, the nodes which are denoted as

FIGURE 1: An example of SaaS software.

FIGURE 2: The tree structure of the interaction relationship of SaaS software.

“seq.,” “branch,” and “loop” represent sequential execution, branch execution, and loop execution, respectively. If the label of the node is “branch,” the value on the subsequent arc means the invocation probability of the corresponding branch, while if the label of the node is “loop,” the value on the subsequent arc represents the execution times of the subsequent node.

2.2. Deployment Scheme and Deployment Resources. A deployment scheme describes the deployment resources that support the operation of SaaS software and the allocation of the service instances in SaaS software to deployment resources. Figure 3 describes a possible deployment scheme for the SaaS software in Figure 1. The deployment resources in this deployment scheme are four deployment nodes, which are marked as n_1 , n_2 , n_3 , and n_4 from top to bottom. Each node is a virtual machine (VM), and the value in the parenthesis means the VM type of node. Each service in SaaS software has one or more instances deployed on different deployment nodes. In this deployment scheme, there is one instance of service S_1 and S_2 , respectively, and two instances of service S_3 , S_4 , and S_5 , respectively. For instance, the instance of S_3 on n_2 is denoted as $s_{3,2}$, and the instance of S_3 on n_4 is denoted as $s_{3,4}$.

2.3. Problem Statement. To facilitate the problem statement, first, we give the definition of the scaling-out operation of SaaS software as follows.

Definition 1 (Scaling-Out Operation of SaaS Software). The operation of scaling out SaaS software is defined as a five-tuple $SO = \{D_{init}, SS, S_i, Act(s_{i,t}, n_t), D_{new}\}$, where we have the following:

D_{init} is the initial deployment scheme of SS

FIGURE 3: An example of the deployment scheme for the SaaS software.

SS is the SaaS software

S_i is one of the services that constitute SS

$s_{i,t}$ is the new instance of S_i generated after the scaling-out operation

n_t is a deployment node in D_{init}

$Act(s_{i,t}, n_t)$ is a function of the scaling-out operation, which generates the new instance $s_{i,t}$ and deploys $s_{i,t}$ on n_t

D_{new} is the new deployment scheme generated after the scaling-out operation

According to Definition 1, the process of scaling out software can be regarded as a series of scaling-out operations. Let the response time of SS before scaling-out operations be $R_{init}(SS)$, and let the response time after scaling-out operations be $R_{new}(SS)$. The problem and goal of this study are to analyse the impact of scaling-out operations on response time of SaaS software and find an effective approach to obtain a better response time $R_{new}(SS)$ comparing with $R_{init}(SS)$ through scaling-out operations without additional deployment nodes.

3. Impact Analysis of Scaling Out SaaS Software on Response Time

For the purpose of analysing the impact of scaling out SaaS software on response time, an approach to evaluate SaaS software response time is needed. In this section, first, we will briefly introduce the evaluation method of response time by leveraging a queueing performance model. With the evaluation method, we will focus on analysing the impact of scaling-out operations on response time of the software. Finally, based on the analysis conclusions, we will further derive the algorithm of scaling out software for response time improvement. The flow of our work is also shown in Figure 4.

3.1. Evaluation of SaaS Software Response Time. Since the SaaS software consists of a series of services that are deployed on multiple deployment nodes, the response time of SaaS

software can be calculated by aggregating the response time of each service on each deployment node [7]. Let n_k be a deployment node, and the VM type of n_k is v_{n_k} whose computing power is $CP(v_{n_k})$. S_i is any service that composes the SaaS software SS, and there is at least one instance of service S_i deployed on n_k (denoted as $s_{i,k}$). According to the Utilization Law [8], the utilization of service instance $s_{i,k}$ to its deployment node n_k can be calculated by the throughput of $s_{i,k}$ and the actual time requirement of $s_{i,k}$ processing a request on n_k as follows:

$$U(n_k, s_{i,k}) = \frac{T(n_k, s_{i,k}) \times TR(S_i)}{CP(v_{n_k})} = \frac{q_i \times p_{i,k} \times T \times TR(S_i)}{CP(v_{n_k})}, \quad (1)$$

where $T(n_k, s_{i,k})$ is the throughput of instance $s_{i,k}$, T is the throughput of SS, q_i is the invocation probability of service S_i , $p_{i,k}$ is the workload proportion of instance $s_{i,k}$, i.e., the proportion of the workload assigned for $s_{i,k}$ to the workload of S_i , and $TR(S_i)$ is the time requirement of S_i on a node with unit computing resource. Then, the utilization of n_k (denoted as $U(n_k)$) can be calculated by the sum of the utilization of each service instance deployed on n_k :

$$U(n_k) = \sum_{S_i \in DS(n_k)} \frac{q_i \times p_{i,k} \times T \times TR(S_i)}{CP(v_{n_k})}, \quad (2)$$

where $DS(n_k)$ is the service type set of service instances that are deployed on n_k .

According to existing research on performance based on queueing theory, each deployment node n_k can be modelled as an M/M/1 queue with a service centre that has $CP(v_{n_k})$ computing power and serves the requests in the processor-sharing (PS) discipline, and the interarrival time of requests and the service time requirement follow exponential distribution [9, 10]. Then, the response time of each service instance $s_{i,k}$ on n_k (denoted as $R(n_k, s_{i,k})$) can be calculated from the request arrival rate of $s_{i,k}$ on n_k (denoted as $\lambda(n_k, s_{i,k})$) and the average request processing rate of $s_{i,k}$ on n_k (denoted as $\mu(n_k, s_{i,k})$) as follows [11]:

$$R(n_k, s_{i,k}) = \frac{1}{\mu(n_k, s_{i,k}) - \lambda(n_k, s_{i,k})}, \quad (3)$$

where $\lambda(n_k, s_{i,k})$ is approximately equal to $T(n_k, s_{i,k})$ in the balanced status and $\mu(n_k, s_{i,k})$ can be calculated by dividing the amount of computing resources of n_k available to the service instance $s_{i,k}$ by the time requirement of service S_i processing a request with unit computing resource as follows:

$$\mu(n_k, s_{i,k}) = \frac{CP(v_{n_k}) \times (1 - U(n_k) + U(n_k, s_{i,k}))}{TR(S_i)}. \quad (4)$$

Therefore, according to equations (1), (2), and (4), equation (3) can be expressed as follows:

$$R(n_k, s_{i,k}) = \frac{TR(S_i)}{CP(v_{n_k}) \times (1 - U(n_k))}. \quad (5)$$

FIGURE 4: The flow diagram of our work.

Then, the response time of service S_i (denoted as $R(S_i)$) can be calculated according to the response times of the corresponding service instances as follows:

$$R(S_i) = \sum_{n_k \in \text{DN}(S_i)} p_{i,k} \times R(n_k, s_{i,k}), \quad (6)$$

where $\text{DN}(S_i)$ is the set of deployment nodes on which the instances of S_i are deployed.

Finally, the response time of the entire software can be calculated by aggregating the response time of each service with the corresponding invocation probability as follows:

$$R(\text{SS}) = \sum_{k=1}^N \frac{\sum_{S_i \in \text{DS}(n_k)} q_i \times p_{i,k} \times \text{TR}(S_i)}{\text{CP}(v_{n_k}) - T \times \sum_{S_i \in \text{DS}(n_k)} q_i \times p_{i,k} \times \text{TR}(S_i)}, \quad (7)$$

where N is the total number of deployment nodes.

After scaling out the software, the response time of the software in the newly generated deployment scheme can also be calculated by equation (7). Therefore, based on equation (7), we can further analyse the relationship between scaling-out operations of software and variation of response time in the next subsection.

3.2. Analysing the Variation of Response Time with the Scaling-Out Operation. For ease of description, the following definitions are given first.

Definition 2 (Response-Time Contribution of Service Instance). Let $s_{i,k}$ (deployed on deployment node n_k) be an instance of service S_i in SaaS software SS, and then, the response time of $s_{i,k}$ is $R(n_k, s_{i,k})$, which is called the response time contribution of service instance $s_{i,k}$ to response time of SS.

Definition 3 (Response-Time Contribution of the Deployment Node). Let $\text{DS}(n_k)$ be the service type set of service instances deployed on deployment node n_k , and then, the response time contribution of deployment node n_k (denoted as $R(n_k)$) means the sum of the response time contribution of each service instance on n_k , which can be calculated as follows:

$$R(n_k) = \frac{\sum_{S_i \in \text{DS}(n_k)} q_i \times p_{i,k} \times \text{TR}(S_i)}{\text{CP}(v_{n_k}) - T \times \sum_{S_i \in \text{DS}(n_k)} q_i \times p_{i,k} \times \text{TR}(S_i)}. \quad (8)$$

Based on Definition 3, the response time of the SaaS software can be seen as the aggregation of the response time contribution of each deployment node in a deployment scheme, as well as the response time after scaling out the software. Assume that, in deployment scheme D_{init} , service S_i has n instances deployed on nodes $n_{k1}, n_{k2}, \dots, n_{km}, \dots, n_{kn}$, respectively (the instances are denoted as $s_{i,k1}, s_{i,k2}, \dots, s_{i,km}, \dots, s_{i,kn}$, correspondingly, and we call them existing instances of S_i). Now, we perform a scaling-out operation according to Definition 1, and then, a new service

instance of S_i is deployed on node n_t (this instance is denoted as $s_{i,t}$) and a new deployment scheme D_{new} is generated. Comparing the response time contribution of nodes in D_{new} with that in D_{init} , we can see that only the response time contribution of the nodes on which the instances of S_i are deployed are varied, i.e., the impact of the scaling-out operation on response time is just from the variation of response time contribution of $n_{k1}, n_{k2}, \dots, n_{km}, \dots, n_{kn}$ and n_t . Hence, in the following part, we will focus on analysing the impact of the scaling-out operation on response time contribution of these nodes.

3.2.1. The Impact of the Scaling-Out Operation on Response Time Contribution of Nodes $n_{k1}, \dots, n_{km}, \dots, n_{kn}$. In deployment scheme D_{init} , based on equation (8), the response time contribution of any of these nodes n_{km} (denoted as $R_{\text{init}}(n_{km})$) can be calculated as follows:

$$R_{\text{init}}(n_{km}) = \frac{\sum_{S_u \in \text{DS}(n_{km})} q_u \times p_{u,km} \times \text{TR}(S_u)}{\text{CP}(v_{n_{km}}) - T \times \sum_{S_u \in \text{DS}(n_{km})} q_u \times p_{u,km} \times \text{TR}(S_u)}, \quad (9)$$

where S_u represents the service type of instance deployed on node n_{km} including S_i , and equation (9) can be expressed as follows:

$$R_{\text{init}}(n_{km}) = \frac{\sum_{S_o \in \text{DS}(n_{km}) \cap S_o \neq S_i} q_o \times p_{o,km} \times \text{TR}(S_o) + q_i \times p_{i,km} \times \text{TR}(S_i)}{\text{CP}(v_{n_{km}}) - T \times \sum_{S_o \in \text{DS}(n_{km}) \cap S_o \neq S_i} q_o \times p_{o,km} \times \text{TR}(S_o) - T \times q_i \times p_{i,km} \times \text{TR}(S_i)}, \quad (10)$$

where S_o represents the service type (other than S_i) of instance deployed on node n_{km} , q_o is the invocation probability of S_o , $p_{o,km}$ is the workload proportion of instance of S_o deployed on n_{km} , and $\text{TR}(S_o)$ is the time requirement of S_o .

After deploying $s_{i,t}$ on n_t , since in deployment scheme D_{new} the instance number of service S_i changed from n to $n+1$, the workload proportion of instance $s_{i,km}$ on n_{km} (i.e., $p_{i,km}$) will decrease (we denoted it as $p'_{i,km}$ in D_{new}):

$$p'_{i,km} < p_{i,km}. \quad (11)$$

Therefore, similar to equation (10), in D_{new} the response time contribution of node n_{km} (denoted as $R_{\text{new}}(n_{km})$) can be expressed as follows:

$$R_{\text{new}}(n_{km}) = \frac{\sum_{S_o \in \text{DS}(n_{km}) \cap S_o \neq S_i} q_o \times p_{o,km} \times \text{TR}(S_o) + q_i \times p'_{i,km} \times \text{TR}(S_i)}{\text{CP}(v_{n_{km}}) - T \times \sum_{S_o \in \text{DS}(n_{km}) \cap S_o \neq S_i} q_o \times p_{o,km} \times \text{TR}(S_o) - T \times q_i \times p'_{i,km} \times \text{TR}(S_i)}. \quad (12)$$

Comparing equation (12) with equation (10) and according to equation (11), we can obtain the conclusion that $R_{\text{new}}(n_{km}) < R_{\text{init}}(n_{km})$, which means that the response time contribution of node n_{km} will decrease after the scaling-out operation.

Similarly, we can obtain the conclusion that the response time contribution of nodes $n_{k1}, n_{k2}, \dots, n_{km}, \dots, n_{kn}$ will all decrease after the scaling-out operation.

We combine the example of SaaS software shown in Figure 1 and the initial deployment scheme shown in Figure 3 to further explain the abovementioned analysis conclusion. For instance, we consider a scaling-out operation $\text{SO} = \{D_{\text{init}}, \text{SS}, S_5, \text{Act}(s_{5,1}, n_1), D_{\text{new}}\}$, i.e., adding an instance for service S_5 ($s_{5,1}$) and deploying it to node n_1 through the scaling-out operation SO . Since before the scaling-out operation there exist two instances of S_5 , i.e., $s_{5,2}$ on node n_2 and $s_{5,4}$ on node n_4 , according to the

abovementioned analysis, the response time contribution of node n_2 and n_4 will decrease after the scaling-out operation.

3.2.2. The Impact of the Scaling-Out Operation on Response Time Contribution of Node n_t . Based on equation (8), in deployment scheme D_{init} , the response time contribution of node n_t (denoted as $R_{\text{init}}(n_t)$) can be calculated as follows:

$$R_{\text{init}}(n_t) = \frac{\sum_{S_v \in \text{DS}(n_t)} q_v \times p_{v,t} \times \text{TR}(S_v)}{\text{CP}(v_{n_t}) - T \times \sum_{S_v \in \text{DS}(n_t)} q_v \times p_{v,t} \times \text{TR}(S_v)}. \quad (13)$$

After deploying $s_{i,t}$ on n_t , the response time contribution of node n_t in D_{new} (denoted as $R_{\text{new}}(n_t)$) can be calculated as follows:

$$R_{\text{new}}(n_t) = \frac{\sum_{S_v \in \text{DS}(n_t) \cap S_v \neq S_i} q_v \times p_{v,t} \times \text{TR}(S_v) + q_i \times p_{i,t} \times \text{TR}(S_i)}{\text{CP}(v_{n_t}) - T \times \sum_{S_v \in \text{DS}(n_t) \cap S_v \neq S_i} q_v \times p_{v,t} \times \text{TR}(S_v) - T \times q_i \times p_{i,t} \times \text{TR}(S_i)}. \quad (14)$$

Comparing equation (14) with (13), we can obtain the conclusion that $R_{\text{new}}(n_t) > R_{\text{init}}(n_t)$, which means that the response time contribution of node n_t will increase after the scaling-out operation.

We also combine the example in Figure 1 to further explain the abovementioned analysis conclusion. We still consider the scaling-out operation $SO = \{D_{\text{init}}, SS, S_5, \text{Act}(s_{5,1}, n_1), D_{\text{new}}\}$. Since the operation adds a new instance $s_{5,1}$ to node n_1 , according to the abovementioned analysis, the response time contribution of node n_1 will increase after the operation.

To conclude, we can see that a scaling-out operation will decrease the response time contribution of the nodes where the instances of S_i already exist before the operation, which makes a positive influence on the response time improvement of the software, while it will also increase the response time contribution of the node where the new instance of S_i is deployed, which makes a negative influence on the response time improvement of the software. If a SaaS provider can perform a series of scaling-out operations to make the positive influence larger than the negative influence, the response time of the software can be effectively improved.

In the next subsection, we will further give an approach of scaling out software for response time improvement based on the abovementioned analysis conclusions.

3.3. Response Time Improvement Based on Scaling Out Software. According to the analysis in Section 3.2, it can be seen that scaling out SaaS software without additional deployment resources will lead to both positive and negative influences on response time improvement of SaaS software. Therefore, to effectively improve the response time by scaling out software, we should determine a series of appropriate scaling-out operations that can make a positive influence on the response time of the software larger than the negative influence. For ease of description, first, we give the following theorem.

Theorem 1. Let S_i be one of the services that constitute the SaaS software SS, and there are n existing instances of S_i deployed on deployment nodes $n_{k1}, \dots, n_{km}, \dots, n_{kn}$ (denoted as $s_{i,k1}, s_{i,k2}, \dots, s_{i,km}, \dots, s_{i,kn}$) in deployment scheme D_{init} . We execute a scaling-out operation $SO = \{D_{\text{init}}, SS, S_i, \text{Act}(s_{i,t}, n_t), D_{\text{new}}\}$, and then, the response time of SS can be improved if the judgment metric $\Delta J(SO) > 0$, where

$$\Delta J(SO) = \frac{p_{i,km} - p'_{i,km}}{p_{i,t}} \times \sum_{m=1}^n \frac{CP(v_{n_t}) \times (1 - U_{D_{\text{init}}}(n_t)) \times (1 - U_{D_{\text{new}}}(n_t))}{CP(v_{n_{km}}) \times (1 - U_{D_{\text{init}}}(n_{km})) \times (1 - U_{D_{\text{new}}}(n_{km}))} - 1, \quad (15)$$

in which $CP(v_{n_{km}})$ and $CP(v_{n_t})$ are the computing power of node n_{km} and n_t , respectively, $U_{D_{\text{init}}}(n_{km})$, $U_{D_{\text{init}}}(n_t)$, $U_{D_{\text{new}}}(n_{km})$, and $U_{D_{\text{new}}}(n_t)$ are utilizations of node n_{km} and n_t in D_{init} and D_{new} , respectively, $p_{i,km}$ and $p'_{i,km}$ are the workload proportions of $s_{i,km}$ in D_{init} and D_{new} , respectively, and $p_{i,t}$ is the workload proportion of $s_{i,t}$ in D_{new} .

Proof. Let $R_{\text{init}}(SS)$ and $R_{\text{new}}(SS)$ be the response time of SS in D_{init} and D_{new} , respectively. When the response time is improved after the scaling-out operation, the following inequality should be satisfied:

$$R_{\text{new}}(SS) < R_{\text{init}}(SS). \quad (16)$$

Let $R_{\text{init}}(n_{km})$ and $R_{\text{new}}(n_{km})$ be the response time contribution of node n_{km} in D_{init} and D_{new} , respectively, and

let $R_{\text{init}}(n_t)$ and $R_{\text{new}}(n_t)$ be the response time contribution of node n_t in D_{init} and D_{new} , respectively. According to the analysis conclusions in Section 3.2, since the response time contribution of nodes other than $n_{k1}, \dots, n_{km}, \dots, n_{kn}$ and n_t is constant, inequality (16) can be transformed into

$$R_c + \sum_{m=1}^n R_{\text{new}}(n_{km}) + R_{\text{new}}(n_t) < R_c + \sum_{m=1}^n R_{\text{init}}(n_{km}) + R_{\text{init}}(n_t), \quad (17)$$

where R_c represents the sum of response time contribution of all nodes except $n_{k1}, \dots, n_{km}, \dots, n_{kn}$ and n_t . Based on equations (2), (9), and (12)–(14), inequality (17) can be transformed into

$$\frac{p_{i,km} - p'_{i,km}}{p_{i,t}} \times \sum_{m=1}^n \frac{CP(v_{n_t}) \times (1 - U_{D_{\text{init}}}(n_t)) \times (1 - U_{D_{\text{new}}}(n_t))}{CP(v_{n_{km}}) \times (1 - U_{D_{\text{init}}}(n_{km})) \times (1 - U_{D_{\text{new}}}(n_{km}))} - 1 > 0. \quad (18)$$

Let

$$\Delta J(SO) = \frac{p_{i,km} - p'_{i,km}}{p_{i,t}} \times \sum_{m=1}^n \frac{CP(v_{n_t}) \times (1 - U_{D_{\text{init}}}(n_t)) \times (1 - U_{D_{\text{new}}}(n_t))}{CP(v_{n_{km}}) \times (1 - U_{D_{\text{init}}}(n_{km})) \times (1 - U_{D_{\text{new}}}(n_{km}))} - 1. \quad (19)$$

Then, we can get the conclusion that the response time will be improved after the scaling-out operation if $\Delta J(\text{SO}) > 0$ is satisfied. Therefore, the theorem holds.

We further illustrate Theorem 1 with the example in Figure 1 and the scaling-out operation $\text{SO} = \{D_{\text{init}}, S_5, \text{Act}(s_{5,1}, n_1), D_{\text{new}}\}$. According to Theorem 1, we can calculate the judgment metric $\Delta J(\text{SO})$ by equations (2) and (15). If $\Delta J(\text{SO}) > 0$, it indicates that the scaling-out operation SO can make a positive influence on the response time improvement larger than the negative influence, i.e., SO is a feasible operation which can improve the response time of the software effectively, while if $\Delta J(\text{SO}) \leq 0$, it indicates that SO cannot improve the response time of the software.

Hence, for the given SaaS software, we perform a series of scaling-out operations that satisfy Theorem 1, and then, the response time of the software can be effectively improved. Based on the abovementioned analysis conclusions, Algorithm 1 presents the main process of scaling out SaaS software for improving response time. The algorithm searches multiple feasible scaling-out operations in order to constitute an operation series. Besides, the algorithm will find a relatively optimal operation series through multiple iterations.

The time complexity of Algorithm 1: the time complexity of the proposed algorithm depends on the number of searching the operation series and the number of searching the feasible operations that constitute the operation series. If the number of searching the operation series is i and for each operation series the number of searching the feasible operations is l , then the time complexity of the algorithm is $O(il)$. $\square$

4. Experiments and Analysis

In this section, we conduct experiments with a practical case to verify the theoretical analysis conclusions in the previous section.

4.1. Experimental Setup. The experimental case is a travel planning system, which is composed of several types of services (the structure is shown in Figure 5). Service S_1 counts the most popular places. Service S_2, S_3 , and S_4 recommend a set of travel places for customers according to different conditions. Service S_2 selects travel places within the most popular places based on the acceptable price of customer. Service S_3 selects travel places based on the place popularity. Service S_4 selects travel places both considering the price and the popularity of places. The invocation probabilities of S_2, S_3 , and S_4 are 0.3, 0.3, and 0.4, respectively. Service S_5 plans the best travel route for the trip, and service S_6 arranges the luggage for customers.

The deployment nodes for the experimental case are provided by an experimental environment implemented by the server equipped with Intel Core i5-6500 CPU @ 3.60 GHz and 16 GB DDR4 RAM. Each node is a VM running Ubuntu Server 14.04 LTS. The initial deployment scheme (denoted as D_{init}) of the experimental case is shown in Figure 6, which presents the distribution of service instances in the experimental case on these nodes.

4.2. Analysis and Verification of the Impact of the Scaling-Out Operation on Response Time. In this section, based on the initial deployment scheme D_{init} of the experimental case, we select the services in the experimental case to conduct three trials to analyse and verify the positive and negative influences of performing scaling-out operations on response time. In each trial, we execute a scaling-out operation by adding a new instance of S_4 or S_5 and deploy it on one of the nodes. The specific scaling-out operations are shown in Table 1. Then, we obtain the positive and negative influences of each scaling-out operation with our analysis method. In addition, during the experiments, the workload of service is assigned to each instance on average before and after scaling-out operations.

The experimental results are shown in Table 2. In the table, R_{init} and R_{new} are the response time of the experimental case before and after scaling-out operations, respectively. The positive influence represents the decreased value of response time contribution of nodes where existing instances of S_4 and S_5 are deployed after scaling-out operations, while the negative influence represents the increased value of response time contribution of nodes where new instances of S_4 and S_5 are deployed after scaling-out operations.

As we can see from the results, the scaling-out operation in trial 1 makes a negative influence larger than the positive influence, which means that the response time after the scaling-out operation becomes longer than before, so it is not a feasible scaling-out operation, while the scaling-out operations in trials 2 and 3 can make a negative influence less than the positive influence, so they are feasible scaling-out operations for improving the response time of the experimental case. Besides, the positive influences of scaling-out operations in trials 2 and 3 are the same, and it is because the positive influences all come from node n_3 , and in trials 2 and 3, the changes of the workload proportion of instance $s_{5,3}$ deployed on n_3 are the same due to the workload-balancing strategy.

Moreover, Table 2 also presents the value of the judgment metric ΔJ mentioned in Theorem 1 for each scaling-out operation. When $\Delta J > 0$ (in trials 2 and 3), the entire response time of the experimental case is reduced after executing the corresponding scaling-out operation, and vice versa (in trial 1). Therefore, the judgment metric ΔJ can correctly reflect whether the corresponding scaling-out operation can improve the response time of the experimental case, which can also verify the effectiveness of Theorem 1. Moreover, comparing trials 2 and 3, the larger the value of ΔJ , the better the improvement in response time, which illustrates that ΔJ can also reflect the improvement degree in response time.

4.3. Analysis and Verification of the Algorithm of Scaling Out Software. In this section, we conduct experiments to further verify the effectiveness of the proposed algorithm in response time improvement through scaling out the experimental case without additional deployment nodes. For the purpose of demonstrating the effectiveness of the algorithm

Input: SaaS Software SS, initial deployment scheme D_{init}

Output: a series of scaling-out operations $SO_Series = [SO_1, \dots, SO_l, \dots, SO_x]$, which makes the response time of SS better than in D_{init}

```

(1) while  $i \leq \maxIteration$  do
(2)  $D_{old} \leftarrow D_{init}$ 
(3)  $operationSearchFlag \leftarrow true$ 
(4)  $l \leftarrow 1$ 
(5) while  $operationSearchFlag = true$  do
(6)   Select a type of service  $S_i$  in SS
(7)   Construct a scaling-out operation  $SO_l = \{D_{old}, SS, S_i, Act(s_{i,t}, n_t), D_{new}\}$  /*The operation will add an instance  $s_{i,t}$  for  $S_i$  and
      deploy it on node  $n_t$ */
(8)   Calculate the judgment metric  $\Delta J(SO_l)$  for  $SO_l$  /*Using equations (2) and (15)*/
(9)   if  $\Delta J(SO_l) > 0$  do
(10)     $SO\_Series_{tmp}[l] \leftarrow SO_l$  /*Add the operation  $SO_l$  to the operation series  $SO\_Series_{tmp}$ */
(11)     $l \leftarrow l + 1$ 
(12)     $D_{old} \leftarrow D_{new}$ 
(13)  end if
(14)  if there are other possible scaling-out operations do
(15)     $operationSearchFlag \leftarrow true$ 
(16)  else
(17)     $operationSearchFlag \leftarrow false$ 
(18)  end if
(19) end while
(20) Calculate  $R(SS)$  after executing operation series  $SO\_Series_{tmp}$  /*Using equation (7)*/
(21) if  $R(SS) < R_{best}(SS)$  do
(22)    $SO\_Series \leftarrow SO\_Series_{tmp}$  /*Update the optimal operation series  $SO\_Series$  with current series  $SO\_Series_{tmp}$ */
(23)    $R_{best}(SS) \leftarrow R(SS)$ 
(24) end if
(25)  $i \leftarrow i + 1$ 
(26) end while
(27) return  $SO\_Series$ 

```

ALGORITHM 1: Scaling out SaaS software for response time improvement.

FIGURE 5: The structure of the experimental case.

FIGURE 6: The initial deployment scheme of the experimental case.

TABLE 1: Scaling-out operations in trials.

Trial	Scaling-out operation
1	$SO_1 = \{D_{init}, SS, S_4, Act(s_{4,1}, n_1), D_{new1}\}$
2	$SO_2 = \{D_{init}, SS, S_5, Act(s_{5,1}, n_1), D_{new2}\}$
3	$SO_3 = \{D_{init}, SS, S_5, Act(s_{5,2}, n_2), D_{new3}\}$

on the experimental case in different configurations of deployment resources, we conduct three groups of experiments. In group 1, we use the configuration of deployment resources shown in Figure 6, and based on group 1, we generate the configurations in group 2 and group 3, as shown in Table 3.

During the experiments, for each time, we select a type of service to perform a feasible scaling-out operation according to Theorem 1. We iterate the abovementioned step until there is no possible scaling-out operation to improve the response time and then terminate the experiments. The first and final deployment schemes are denoted as D_{init} and D_{final} . To compare the effectiveness of our algorithm in different groups, for each group, we have recorded the response times of the experimental case under D_{init} and D_{final} , including the response times obtained from our analysis method (denoted as $R_a(D_{init})$ and $R_a(D_{final})$, respectively) and the response

TABLE 2: Experimental results of verifying the impact of scaling-out operations on response time.

Trial	$R_{init}(s)$	$R_{new}(s)$	Positive influence (s)	Negative influence (s)	ΔJ
1	0.8110	0.9209	0.0203	0.1303	-0.8442
2	0.8110	0.6940	0.3230	0.2060	0.5677
3	0.8110	0.5224	0.3230	0.0344	8.3780

TABLE 3: Experimental groups of verifying the algorithm of scaling out software.

Group	Computing power of each node				The number of nodes	Total computing power
	n_1	n_2	n_3	n_4		
1	1	2	1	—	3	4
2	1	2	2	—	3	5
3	1	2	1	1	4	5

times actually measured from the experimental environment (denoted as $R_m(D_{init})$ and $R_m(D_{final})$, respectively), as shown in Figure 7.

The improvement degree in response time brought by the algorithm (calculated by $(R_m(D_{init}) - R_m(D_{final}))/R_m(D_{init}) \cdot 100\%$) in the three experimental groups is 45.5%, 20.3%, and 27.4%, respectively. The average relative error of R_a compared to R_m (calculated by $(|R_a - R_m|)/R_m \cdot 100\%$) is generally less than 8%, and according to the experimental results, the accuracy is acceptable for finding feasible scaling operations in order to improve the response time of the experimental case. Specifically, when the initial response time of the experimental case is relatively long (group 1 and group 3), the algorithm can bring a relatively large improvement degree of response time. Even if the initial response time in group 2 is not as long as that in the other two groups, the response time is further improved by scaling out the experimental case. Therefore, the experiments illustrate that the conclusions of impact analysis and the proposed algorithm can help improve the response time effectively by scaling out software in different configurations of deployment resources, even without additional deployment nodes.

4.4. Analysis of the Statistical Significance of Our Method in Improving Response Time. With the aim of verifying whether the effectiveness of our method in improving the response time of the experimental case is statistically significant, in this section, we further conduct more simulation experiments and perform statistical hypothesis testing to make extensive evaluation of our method. Specifically, based on the experimental case in Figure 5, we conduct three groups of experiments with three different number of initial service instances. Besides, in each group, we also conduct experiments with three different invocation probabilities of services, two different request arrival rates, and two different deployment configurations, i.e., we conduct a total of 36 different experiments.

During the experiments, according to the response times before and after executing scaling-out operations found by

our method, we conduct the paired t -test to verify whether the response times of the experimental case after executing the scaling-out operations are significantly different from the initial response times. We consider two different hypotheses as follows:

- (i) H0: there is no difference between the response times after executing scaling-out operations and the initial response times
- (ii) H1: there is a difference between the response times after executing scaling-out operations and the initial response times

The results of the paired t -test are shown in Table 4, which presents the significance level of the improved response times compared to the initial response times. Concretely, the table shows the number of tests in each group, the degrees of freedom (df), the standard deviation differences (SD) and the mean of differences of the improved response times with the initial response times, the t value, and the p value.

Generally, for a paired t -test, the significance level is defined as $p < 0.05$, i.e., if the p value is less than 0.05, we should reject the null hypothesis (H0) and accept the alternative hypothesis (H1). As shown in Table 4, the p value in each group is less than 0.05, so the null hypothesis (H0) is rejected and the alternative hypothesis (H1) is acceptable, which means that the differences between the improved response times and the initial response times are significant, allowing us to conclude that the effectiveness of our method in improving the response time of the experimental case is statistically significant.

5. Discussion

In the previous section, some experiments have been conducted to evaluate our study, which indicates that the analysis conclusions and the proposed algorithm derived from our study can effectively provide guidance for scaling out SaaS software and improving the response time. Our study is based on the queueing performance model, and its advantage is that it can be used for quantitatively analysing the positive and negative influences of scaling-out operations on the response time of the software in combination with the deployment resources, which can assist in subsequently making decisions for scaling out software with existing deployment resources instead of additional deployment resources. However, we also have identified some limitations. Our current study mainly models the performance of SaaS software, but in the deployment nodes, there also exist some performance interference factors outside of SaaS software, which may not be captured by the model and limit the effectiveness of analysing

FIGURE 7: Results of experiments of verifying the algorithm of scaling out software.

TABLE 4: Results of the paired t -test.

The number of initial service instances	The number of tests	df	SD	Mean of differences	t value	p value
10	12	11	0.0917	0.3323	12.5552	7.30×10^{-08}
15	12	11	0.0786	0.1783	7.8582	7.74×10^{-06}
20	12	11	0.0167	0.1073	22.2610	1.69×10^{-10}

and improving the response time of software. In the future, we can extend our model to take these interference factors into account and further enhance the effectiveness of performance improvement.

6. Related Work

In the field of software performance, there are some works aiming to analyse and improve the performance of software and taking into account the scaling techniques. Vondra et al. [12] developed a new simulation tool based on the queueing model to autoscale VMs in the private cloud. Jiang et al. [13] proposed a novel scheme to autoscale cloud resources for web applications. Bouterse et al. [14] proposed a method to dynamically allocate VMs to provide reserved resources for scalable SaaS applications. These works mainly studied the scaling approaches in combination with resource provisioning. Besides, El Kafhali et al. [15] presented a queueing mathematical model to estimate the amount of VMs required for scaling software in order to satisfy the performance requirement. This work mainly studied the approach of adjusting the number of resources for scaling software.

Some studies also considered the elasticity of resources to guarantee the performance of the scalable software. Salah et al. [16] presented an analytical model that can be used to guarantee proper elasticity for cloud-hosted applications and services, in order to satisfy particular performance requirements. Ghobaei-Arani et al. [17] presented a framework called ControCity for controlling resources elasticity through using buffer management and elasticity management by leveraging the learning automata technique. The aforementioned works mainly considered how to adjust the resources to meet the performance improvement requirement of the scalable software. However, how to specifically adjust and scale the software to improve performance also needs to be further studied.

Zhen et al. [18] proposed a method to automatically scale cloud application instances. Wada et al. [7] presented a method to estimate and optimize the performance of services in a cloud environment by leveraging queueing theory, which considered the strategies of scaling cloud services. These works mainly require adjusting deployment resources while scaling software. Different from the abovementioned works, we aim to improve the response time of SaaS software based on scaling out software without additional adjustment of deployment resources. Therefore, in our work, we analysed both the positive and negative influences of scaling out SaaS software on response time, and based on the above-mentioned foundation, we proposed an algorithm of scaling out SaaS software to improve the response time effectively while saving deployment resources.

7. Conclusions

In this paper, we defined the scaling-out operation of SaaS software and analysed the impact of the scaling-out operation on response time by leveraging queueing theory. The experiments showed that the analysis conclusions can be used to determine whether a scaling-out operation can effectively improve the response time of the software, which can be the basis of making decision for scaling out software. Based on the analysis conclusions, we further proposed an algorithm of scaling out SaaS software for response time improvement. The experiments demonstrated that the response time can be reduced effectively after scaling out the software with the proposed algorithm, which further illustrated that the analysis conclusions obtained from this paper can play a guiding role in scaling out software and improving response time while saving deployment resources. In the future, we will further explore how to apply the analysis conclusions to optimize the performance of SaaS software.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant nos. 62072342, 61672392, and 61373038.

References

- [1] W. Tsai, X. Bai, and Y. Huang, "Software-as-a-Service (SaaS): perspectives and challenges," *Science China Information Sciences*, vol. 57, no. 5, pp. 1–15, 2014.
- [2] B. Steffen, G. Brataas, and S. Lehrig, *Engineering Scalable, Elastic, and Cost-Efficient Cloud Computing Applications*, Springer, Switzerland, Europe, 2017.
- [3] U. N. Bhat, *An Introduction to Queueing Theory: Modeling and Analysis in Applications*, Birkhäuser, Dallas, Texas, 2015.
- [4] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queueing theory model for cloud computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 492–507, 2014.
- [5] V. Cortellessa, A. Di Marco, and P. Inverardi, *Model-Based Software Performance Analysis*, Springer-Verlag, Berlin, Germany, 2011.
- [6] P. Pasqualina, "Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability and performance tradeoff," *Journal of Systems and Software*, vol. 86, no. 3, pp. 624–648, 2013.
- [7] H. Wada, J. Suzuki, Y. Yamano, and K. Oba, "Evolutionary deployment optimization for service-oriented clouds," *Software: Practice and Experience*, vol. 41, no. 5, pp. 469–493, 2011.
- [8] D. A. Menasce, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*, Prentice Hall Professional, New Jersey, USA, 2004.
- [9] H. D. Lee, "An efficient VM-level scaling scheme in an IaaS cloud computing system: a queueing theory approach," *International Journal of Contents*, vol. 13, no. 2, pp. 29–34, 2017.
- [10] A. N. Dudin, S. A. Dudin, O. S. Dudina, and K. E. Samouylov, "Analysis of queueing model with processor sharing discipline and customers impatience," *Operations Research Perspectives*, vol. 5, pp. 245–255, 2018.
- [11] W. C. . Chan, *An Elementary Introduction to Queueing Systems*, World Scientific, Singapore, Asia, 2014.
- [12] T. Vondra and J. Šedivý, "Cloud autoscaling simulation based on queueing network model," *Simulation Modelling Practice and Theory*, vol. 70, pp. 83–100, 2017.
- [13] J. Jiang, "Optimal Cloud Resource Auto-Scaling for Web Applications," in *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pp. 58–65, Delft, Netherlands, May 2013.
- [14] B. Bouterse and H. Perros, "Performance analysis of the reserve capacity policy for dynamic VM allocation in a SaaS environment," *Simulation Modelling Practice and Theory*, vol. 93, pp. 293–304, 2019.
- [15] S. El Kafhali and K. Salah, "Performance analysis of multi-core VMs hosting cloud SaaS applications," *Computer Standards & Interfaces*, vol. 55, pp. 126–135, 2018.
- [16] K. Salah, K. Elbadawi, and R. Boutaba, "An analytical model for estimating cloud resources of elastic services," *Journal of Network and Systems Management*, vol. 24, no. 2, pp. 285–308, 2016.
- [17] M. Ghobaei-Arani, A. Souri, T. Baker, and A. Hussien, "ControCity: an autonomous approach for controlling elasticity using buffer Management in Cloud Computing Environment," *IEEE Access*, vol. 7, pp. 106912–106924, 2019.
- [18] X. Zhen, Q. Chen, and H. Luo, "Automatic scaling of internet applications for cloud computing services," *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1111–1123, 2014.