Future Generation Computer Systems 107 (2020) 551–563

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Integrating recurrent neural networks and reinforcement learning for dynamic service composition

Hongbing Wang [a,*], Jiajie Li [a], Qi Yu [b], Tianjing Hong [a], Jia Yan [a], Wei Zhao [a]

[a] School of Computer Science and Engineering and Key Laboratory of Computer Network and Information Integration, Southeast University, SIPAILOU 2, Nanjing 210096, China
[b] College of Computing and Information Sciences, Rochester Institute of Tech, USA

ARTICLE INFO

Article history:
Received 26 April 2019
Received in revised form 5 December 2019
Accepted 7 February 2020
Available online 15 February 2020

Keywords:
Service composition
QoS prediction
Recurrent neural network
Reinforcement learning

ABSTRACT

In the service oriented architecture (SOA), software and systems are abstracted as web services to be invoked by other systems. Service composition is a technology, which builds a complex system by combining existing simple services. With the development of SOA and web service technology, massive web services with the same function begin to spring up. These services are maintained by different organizations and have different QoS (Quality of Service). Thus, how to choose the appropriate service to make the whole system to deliver the best overall QoS has become a key problem in service composition research. Furthermore, because of the complexity and dynamics of the network environment, QoS may change over time. Therefore, how to adjust the composition system dynamically to adapt to the changing environment and ensure the quality of the composed service also poses challenges. To address the above challenges, we propose a service composition approach based on QoS prediction and reinforcement learning. Specifically, we use a recurrent neural network to predict the QoS, and then make dynamic service selection through reinforcement learning. This approach can be well adapted to a dynamic network environment. We carry out a series of experiments to verify the effectiveness of our approach.

© 2020 Elsevier B.V. All rights reserved.

1. Introduction

The Internet has become an important part of people's daily life. Network technology has also had a great impact on the software development in many application domains. More and more enterprises and organizations deliver their softwares, systems, computing resources, storage resources, and so on in the form of Web services to be used by other users. Therefore, how to effectively integrate various services has become a fundamental research problem. Service composition mainly studies how to produce a system that can meet the complex requirements by combining existing services. It has significant importance in the development of modern software systems. In addition, services that have the same functionalities may be provided by different service providers with different Quality of Service (QoS). QoS is mainly used to denote the non-functional attributes of services, which include price, usability, reliability, response time, reputation, throughput, and so on. Due to the dynamic change of the network environment, the performance fluctuation of the service itself, and the change of users' access patterns, the QoS may also change dynamically over time. Therefore, the service composition methods need to adapt to the dynamic changing environment [1].

Currently the process of service composition is often represented as a workflow. A typical workflow is shown in Fig. 1, where a solid circle represents an abstract service while a hollow circle represents a state node. The abstract service describes the required service functionality, and each abstract service has several specific services that meet the requirements that can be invoked, known as candidate services. Workflow-based service composition focuses on QoS attributes. For each abstract service in the workflow, a proper specific service is determined through QoS attributes during the execution of service composition, and finally an optimal service composition result can be obtained, which can meet user requirements to the greatest extent.

This paper assumes that the workflow which meets the requirement of the user has been built, but each subtask has a number of candidate services with similar functionalities but different QoS to be selected. Because of the dynamic change of the network environment and the change of the service itself, the QoS of each service will fluctuate. The proposed approach combines QoS prediction with reinforcement learning, where the former is used to estimate the QoS and improve the accuracy of service selection under the dynamic environment, while the latter provides self optimization and adaptive ability for the service composition.

* Corresponding author.
E-mail addresses: hbw@seu.edu.cn (H. Wang), qi.yu@rit.edu (Q. Yu).

https://doi.org/10.1016/j.future.2020.02.030
0167-739X/© 2020 Elsevier B.V. All rights reserved.

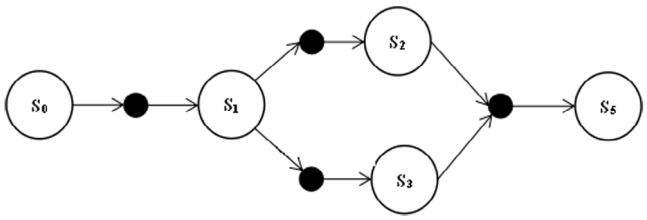


Fig. 1. Service composition workflow.

Table 1
Notations.

ANN	Artificial Neural Network
BPEL	Business Process Execution Language
DWSC-MDP	Dynamic Web Service Composition Markov Decision Process
LSTM	Long Short Term Memory
MDP	Markov Decision Process
QoS	Quality of Service
QP-RL	Q Prediction Reinforcement Learning
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XML	eXtensible Markup Language

This leads to a new adaptive service composition method for QoS-aware service composition that can adapt to the dynamic network environment. Our contributions are summarized as follows:

(1) We apply the time series prediction method to the field of service computing. The QoS prediction method based on a recurrent neural network is developed.
(2) We combine reinforcement learning with QoS prediction and apply them to service composition.
(3) We conduct a series of experiments to verify the method proposed in this paper, which prove its effectiveness in the dynamic environment.

The remainder of this paper is organized as follows. Section 2 describes related works. Section 3 gives a brief description of the relevant theories, including reinforcement learning and time series prediction. Section 4 mainly introduces the service composition model based on the Markov process and a new service composition method based on reinforcement learning and QoS prediction. Section 5 shows the experimental results to verify the method proposed in this paper. Section 6 presents the concluding remarks. Table 1 summarizes the notations used in this paper.

2. Related work

In this section, we review some related works about adaptive service composition, including reinforcement learning, QoS prediction and some other solutions adopted in service composition.

Considering the dynamics and complexity of the network environment, QoS may be constantly changing, so the method of service composition is required to perceive the changes of the environment and adjust the system automatically. This challenge has attracted widespread attention and many solutions have been developed. In [2], the authors use the artificial intelligence method to plan the service composition. When the quality of a service decreases, this method will replace it with a better one. But because only a single service replacement is considered, this method lacks overall control over the entire composite system. It may appear that the overall QoS decreases. In [3,4], the authors consider the problem of service composition optimization as an integer programming problem, and then obtain the optimal solution by using related technology. In [5,6], the authors describe the relationships among various services through a plan map, model the service composition using a plan diagram model, and then search for poor quality services and replace them with a greedy algorithm. In [7], the authors also use integer programming to make global optimization of the service composition. At the beginning, by using skyline technology, the number of candidate services is greatly reduced and thus the difficulty of integer programming is greatly reduced. Similarly, in [8], the authors also use skyline technology to filter services. The basic idea of the above methods is to solve the problem of service composition by integer programming or mixed integer programming. But this kind of methods are only applicable to small-scale problems, and lack self-adaptive ability. So they cannot solve the problem of service composition in a large-scale and dynamic environment. Besides, the evolution algorithms were also used for service selection and composition. In [9], an ant colony optimization was proposed to supported global QoS optimization of service composition. But it is difficult for this method to get rid of the local optimal problem. In [10], the balance between exploration and exploitation was considered, and eagle strategy was used with the whale optimization algorithm for the cloud service composition. this method can avoid the local optimum, but not consider the change of the QoS with time, which affects the adaptability.

In addition, in recent years, machine learning has gradually become a research hotspot in the field of information technology. Some scholars have tried to apply reinforcement learning to service composition. In [11], the authors use the Markov decision process model and reinforcement learning to perform adaptive service composition. In [12], for the efficiency problem of reinforcement learning, the authors use a hierarchical reinforcement learning algorithm to improve the efficiency of service composition. In [13–15], the authors apply the multi-agent technology to service composition. These methods improve the efficiency of service composition and the quality of the service composition result. In [16], a three-layer trust-enabled service composition model was proposed. Taking advantage of fuzzy comprehensive evaluation method, the authors introduced an integrated trust management model. This model could analyze the preferences of users well to obtain better service composition. However, the above algorithms based on reinforcement learning have some adaptability, but they ignore some special problems in service composition. More specifically, services are often deployed in the network environment. The QoS often changes over time due to the dynamic network environment and the possible performance fluctuations of the service itself. Therefore, to obtain the better results of the service composition, the changed quality of the service needs to be predicted effectively. Although the traditional reinforcement learning method has a certain degree of self-adaptive ability, it still cannot accurately estimate the quality of service, which results in the unsatisfactory results of the service composition in a highly dynamic environment.

In the service composition, the environment is dynamic, and the QoS varies with time. Therefore, it is necessary to predict the change of QoS in some way, which can be achieved through QoS prediction. In recent years, a large number of scholars have studied the prediction of QoS. The most common study is collaborative filtering based approaches [17–22]. This kind of methods assume that similar users can get similar QoS from a service. They classify users through some features and provide quality prediction for other similar users by using the historical data produced by a user who has used the service. QoS prediction based on collaborative filtering is more concerned with the QoS differences between different users. But in service composition, the workflow of the whole service composition is designed to

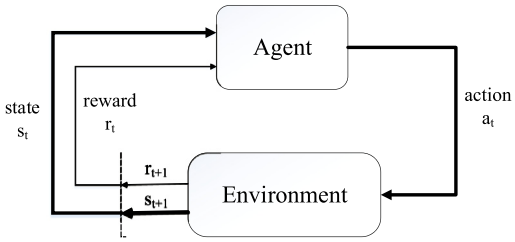


Fig. 2. The basic framework of reinforcement learning.

meet the requirements of a certain user. Each service is called by this user. The quality of service changes is not caused by the change in the location or state of the service caller, but because the network environment at that time has changed. The QoS varies over time, and can be described as a time series data, which is similar to the value of stocks. In this background, some scholars try to apply the time series forecasting technology to the prediction of QoS.

In [23], Rodrigo et al. predict time series of QoS based on the autoregressive integrated moving average (ARIMA). However, the ARIMA model requires time series data to be stable, hence it cannot be applied to scenarios with highly dynamic changes. Recently, with the development of deep learning technology, some researchers have studied the sequence prediction methods based on recurrent neural networks, which have made some certain progress in other fields [24–26]. For example, in [27], RNN has been used for multi-step ahead prediction, which can fit a wider range of data patterns. Overall, deep learning based methods are better at fitting the complex functions compared with the traditional statistical models. Therefore, they can provide better prediction performance when dealing with complicated time series.

3. Preliminaries

In this section, we will introduce some preliminaries about our approach, including reinforcement learning and time series prediction.

3.1. Reinforcement learning

Reinforcement learning is one of the most important machine learning methods. It is good at achieving some purposes in an unknown environment by constantly adjusting the behavior of the agent, for example, to control the robot to find the exit in the maze. Reinforcement learning simulates the exploration behavior of human or other organisms in an unknown environment. In the unknown environment, the agent constantly interacts with the environment, gets feedback, and then adjusts its own behavior. The ultimate goal is to maximize the reward obtained from the environment [28].

The basic framework of reinforcement learning is shown in Fig. 2. Agent executes an action a_t. After execution, the reward value r_{t+1} is obtained from the environment, and then the agent transfers to the new state s_{t+1}. The agent will adjust the action selection according to the reward, and finally form its own action selection strategy, which obtains the maximum reward value.

3.1.1. The input and output of reinforcement learning

According to the basic framework and principles of reinforcement learning, we can describe the input and output of the reinforcement learning algorithm. The input includes the set of possible states, the executable actions under each state, the reward value that can be obtained and which new state can be transferred to by the execution of an action. The output is a strategy that chooses actions based on states, which should maximize the expectation of total reward value.

In fact, the input of reinforcement learning is a Markov decision process (MDP), which is defined as follows:

Definition 1 (*Markov Decision Process (MDP)*)**.** An MDP is a 4-tuple MDP=$\langle S, A(.), P, R\rangle$, where

- S is a finite set of the world states;
- $A(.)$ is a finite set of actions, in which $A(s)$ represents the set of actions that can be executed in state $s \in S$;
- P is the probability distribution function. When an action a is invoked, the world makes a transition from its current state s to a succeeding state s'. The probability for this transition is labeled as $P(s'|s, a)$;
- R is the immediate reward function. When the current state is s, and an action a is selected, then we can get an immediate reward $r = R(s, a)$ from the environment after executing an action.

The output of reinforcement learning is an action selection strategy, expressed as $\pi : S \to A$, that is, to select action $a = \pi(s)$ under state s. The best action selection strategy should maximize the expectation of total reward value. The total reward value of the agent obtained under a certain state action sequence whose starting state is s_t can be described as G_t:

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{1}$$

where r_{t+i} represents the reward obtained when transferring from state s to s' at time $t+i$. γ indicates the discount rate, with $0 \le \gamma \le 1$, which ensures that short-term reward is greater than long-term reward, and the impact of recent reward plays a more important role in the selection of action. The state-value function is defined as the expectation of the random variable G_t:

$$V^{\pi}(s_t) = E^{\pi}[G_t|s_t = s] = E^{\pi}[\sum_{i=0}^{\infty} \gamma^i r_{t+i}|s_t = s] \tag{2}$$

where $E^{\pi}(*)$ denotes the expected value of a random variable under policy π. $V^{\pi}(s_t)$ represents the expectation of the total rewards from the starting state s_t and under the strategy π. Its recursive expression is

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} p(s'|s, a)[r + \gamma V^{\pi}(s')] \tag{3}$$

where $p(s'|s, a)$ represents the probability of transitioning from state s to succeeding state s'. According to the formula (3), the optimal action selection strategy π^* can be expressed as:

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), \quad \forall s \in S \tag{4}$$

Combining with the definition of a Markov decision process, formula (3) and formula (4), the final best action selection strategy can be also expressed as the formula (5).

$$\pi^* = \operatorname{argmax}_{\pi}(\sum_{s'} p(s'|s, \pi(s))[r + \gamma V^{\pi}(s')]) \tag{5}$$

As a result of the above discussion, the input of the reinforcement learning algorithm is a Markov decision process, and the output is a strategy of selecting the action $a = \pi(s)$ under the state s, which can be expressed by formula (5).

3.1.2. Q-learning

Q-learning is a commonly used reinforcement learning algorithm [29]. Its basic idea is to estimate the reward generated by an action with a Q value table, where the Q value is the estimation of the real reward. When the actual value and the estimated value are different after the execution of an action, the Q value table is updated by the difference between the two value. The Q value can gradually approximate to the real value [30]. After a certain number of iterations, the Q value table can have a more accurate estimate of the real reward value.

The Q-learning algorithm uses $Q(s, a)$ to indicate the estimated reward value of action a under the state s. The action selection strategy follows the formula (6).

$$\pi(s) = \arg\max_a Q(s, a), a \in A(s) \tag{6}$$

Under this strategy, the real reward value of the action a based on the state s can be expressed as the formula (7), where s' represents the next state, and the subscript "real" represents the real reward value, while $Q(s, a)$ represents the value in the Q value table to estimate the real Q value:

$$Q_{real}(s, a) = R(s' \mid s, a) + \gamma \max Q(s', a') \tag{7}$$

The value in the Q table will be updated according to formula (8).

$$\begin{aligned} Q(s, a) = {} & Q(s, a) + \alpha(R(s' \mid s, a) \\ & + \gamma \max Q(s', a') - Q(s, a)) \end{aligned} \tag{8}$$

In the Q-learning algorithm, the theoretical action selection follows formula (6). But in the actual learning process, if every time we choose the action with the greatest Q value, it may easily fall into a local optimum. Therefore, in the action selection of the actual Q-learning algorithm, the $\epsilon - greedy$ strategy is executed. That is, the action selection is conducted using formula (6) with probability of ϵ and using random selection with probability $1 - \epsilon$. This strategy can help the reinforcement learning algorithm achieve a certain balance between exploration and experience utilization, and avoid falling into local optimum too early. The Q-learning algorithm is given in Algorithm 1.

Algorithm 1: Q-learning

```
Initialize Q(s, a), parameters, input the MDP model
repeat
  set current state s
  repeat
    Choose a in A(s) based on ε − greedy policy;
    Execute a, get the reward r = R(s' | s, a) and new state s';
    Q(s, a) = Q(s, a) + α(r + γ maxQ(s', a') − Q(s, a))
    s ← s'
  until Achieve a certain termination state
until Convergence
```

3.2. Time series prediction

The time series prediction is to estimate the future outcomes based on the past data. In the field of service computing, because of the dynamic change of the network environment and the performance fluctuation of the service itself, the QoS often changes with time. This change presents a certain regularity. Therefore, the QoS can be regarded as time series data and our goal is to predict the future QoS based on historical data.

It is assumed that a time series data x_t satisfies $x_t = g(T)$, where $g(T)$ embodies the essential rules of the change of x_t with time. The time series prediction method generally believes that the future data can be predicted based on the past data. That is, there exists a function f, which makes $g(t + 1) \approx f(x_t, x_{t-1}, x_{t-2}, \ldots, x_0)$. The goal of time series prediction is to find the function f, so that the future data can be estimated based on the past time series data [31].

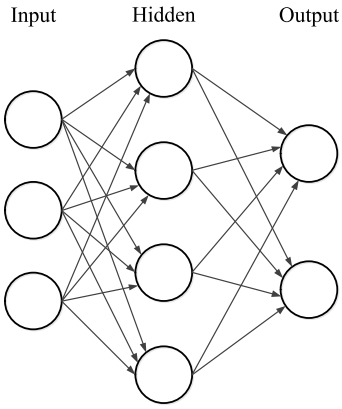


Fig. 3. The basic structure of the neural network.

The classical time series prediction methods include moving average method, exponential smoothing method, autoregressive model and autoregressive summation moving average model [32]. The general principle of these methods is to assume that f is a linear function, and then select a suitable model according to the characteristics of historical data. It establishes a form of $f(\theta)$ with parameter θ, then determines the parameters of the function by analyzing the historical data, and uses this function to predict the future data. Most of these methods are based on short-term data and assume that the objective function f is linear, which have a good effect in short-term prediction or simple time series prediction. However, when the time series changes in a complicated way that cannot be described by linear models, the prediction accuracy of the above models will be relatively low.

3.2.1. Neural network

In recent years, deep learning and neural network technology have developed rapidly. Neural networks connect multiple processing layers to solve complex prediction problems [33]. Because of the ability of neural network to fit complex functions, there are a lot of time series prediction researches based on neural networks. The basic principle of these methods is to establish the neural network $f(\theta)$ first, and then train the network through historical data, adjust its parameters, and finally use the trained neural network to predict the future data [33]. The neural network based method is able to predict complex time series, and can adjust the parameters of neural network by the newly added data. It has better learning ability and adjustment ability [34]. Therefore, this paper adopts a neural network based time series prediction model to predict the future QoS.

The basic structure of the neural network is shown in Fig. 3, which is a neural network containing the input, hidden, and output layers. The link between neurons represents the weight that needs to be multiplied when the data is transmitted. The output of the neuron is finally obtained by the function $y = f(wx + b)$, in which the y is the output of the neuron, f is the activation function of the neuron, w is the weights, b is the bias of neurons, and x represents the data vector of the input layer. Each neuron in a neural network follows the above simple rules, and finally transfers the input of the neural network to the output, which is called the forward propagation of the neural network.

The neural network can be represented as a function with parameters: $y = f(x; \theta)$, where x is input data, y is output data, and θ represents the parameters in the neural network. The

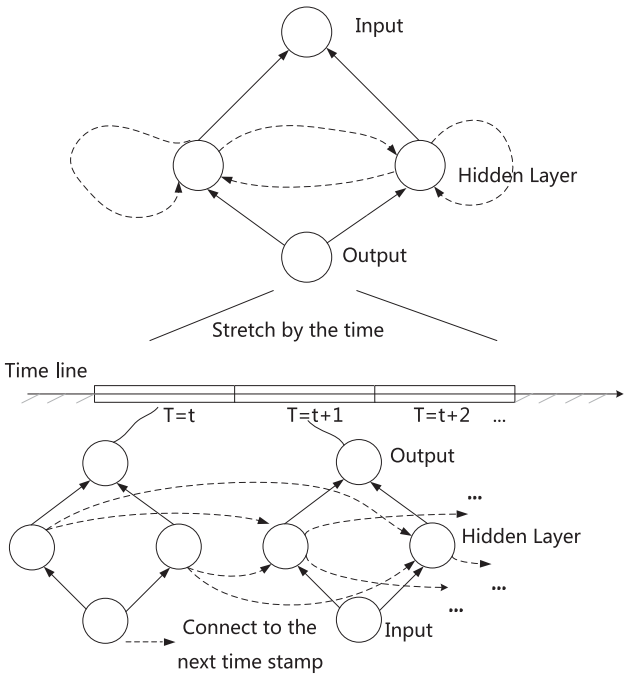


Fig. 4. The structure of a recurrent neural network.

output of an untrained neural network is often unsatisfactory, that is, there is an error between the output and the expected result. With the change of the neural network parameters, the error on a certain set of data may become larger or smaller. Based on this, we can define an error function $L(\theta)$ to describe the degree of the neural network error. Therefore, the training process of neural network is actually to find a set of parameters θ to minimize the value of the error function.

The gradient descent method is often used to solve this problem. The basic principle of the gradient descent method is that the gradient direction is the (locally) fastest changed direction for a function, so the value of the function can be reduced by adjusting the independent variable along the gradient direction. The step is repeated many times until the function value reaches a lowest point. Then, a local minimum value is found. According to this principle, the updating formula of neural network parameters is given below:

$$\theta = \theta - \eta \frac{\partial L(\theta)}{\partial \theta} \tag{9}$$

where η is the learning rate.

3.2.2. Recurrent neural network

Recurrent neural network is a special type of neural networks. On the basis of the basic neural network structure, recurrent neural networks take into account the possible temporal correlation of some data, such as speech recognition, intelligent translation [24,25,35].

The basic structure of a recurrent neural network is shown in Fig. 4 [36]. In traditional neural networks, the input of the hidden layer comes from the input layer only. In the recurrent neural network, the input of hidden layer includes the input layer data and the output of the hidden layer neurons at the last time. This special neural network can also be expanded in time to get the structure displayed in the lower part of Fig. 4.

Hochreite [37] made further modifications to recurrent neural networks and put forward the LSTM neural network. Similar to the human memory, some important information needs to keep for a longer time, while some unimportant information may be

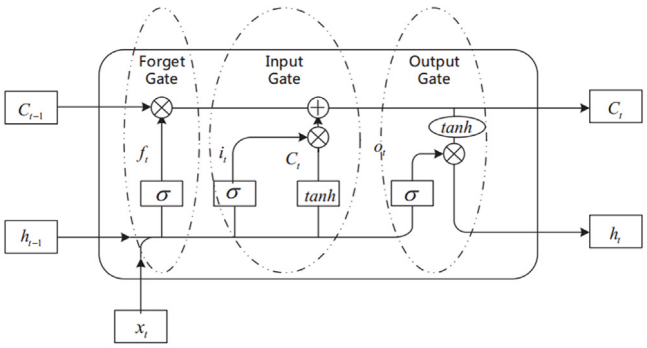


Fig. 5. The LSTM neural network internal structure.

easily forgotten. The LSTM neural network adds some structure to distinguish and process information. It can decide long-term memory or oblivion according to the importance of information. A typical LSTM neural network internal structure is shown in Fig. 5. In the figure, *tanh* and σ represent the neurons that use the corresponding activation function, h represents output, x represents input, and C is the called cell state. The entire internal structure can be divided into three parts, including forget gate, input gate, and output gate, to control the forgotten degree of the original data, the acceptance of the new data and the output data. In the forget gate, the forgotten degree f_t is calculated by the formula (10), which is between 0 and 1. The forget rate is used to control the retention of the original memory.

$$f_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{10}$$

The input gate determines which data to be added to the cell state. It is calculated as:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{11}$$

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_C) \tag{12}$$

Finally, the new cell state is controlled jointly by the calculation results of the forgotten and the input gates:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{13}$$

The output of the LSTM network is determined by the new cell state, the current input and the output of the hidden layer at the last time.

$$h_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) * tanh(C_t) \tag{14}$$

By adding the three gates structure to the recurrent neural network, the LSTM can distinguish and control information, memorize the important information for a long time, and store it in the cell state. In practice, LSTM has shown excellent learning ability and has achieved great success in processing time series data. Therefore, we adopt the LSTM model to learn and predict the time series data of service quality, aiming to improve the accuracy of the QoS estimation in a dynamic environment.

4. Service composition based on QoS prediction and reinforcement learning

In this section, we propose an adaptive service composition method combining QoS prediction and reinforcement learning. At first, a typical scenario of service composition is described to help understand the problem of service composition. In the service composition, we select suitable candidate services through reinforcement learning to maximize the total QoS value of the composite service. Considering the dynamics and complexity of

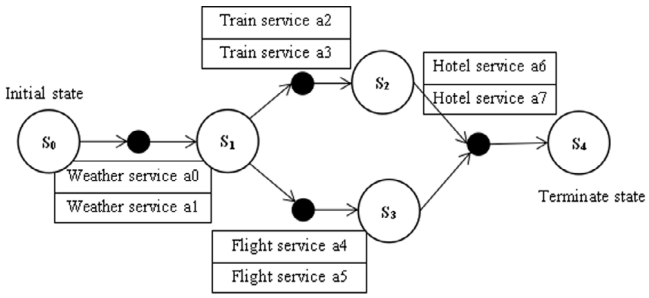


Fig. 6. A simple service composition workflow.

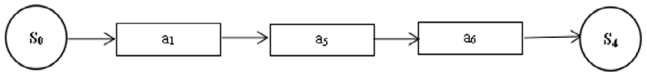


Fig. 7. A possible service composition result.

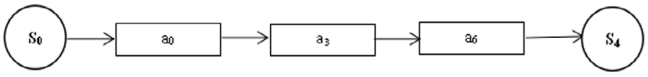


Fig. 8. Another possible service composition result.

the network environment, QoS is always changing, so we introduce the QoS prediction method based on LSTM. Finally, we combine the QoS prediction and reinforcement learning to solve service composition problem, which can improve the quality of the composite service in dynamic environment.

4.1. A service composition scenario

A typical service composition flow is shown in Fig. 6. For each abstract service, we need to determine its specific services, and ultimately formulate a specific service composition workflow. For example, based on the service composition workflow in Fig. 6, the possible service composition results are shown in Fig. 7 and Fig. 8. It is straightforward to see that as the number of candidate services increases and the complexity of the service composition workflow increases, the number of possible service composition results will dramatically increase. The challenge of service composition is to check a large candidate space and get the optimal result that maximizes the reward value.

4.2. Service composition model

According to the problem described in the previous section, we will propose a service composition model based on the workflow. The workflow of the service composition can be regarded as a Markov decision process. Based on the definition of Markov decision process, the Markov decision process model of Web service composition in a dynamic environment is defined below:

Definition 2 (*Dynamic Web Service Composition MDP (DWSC-MDP)*)**.** A DWSC-MDP is a 6-tuple DWSC-MDP=$\langle S, S_0, S_r, A(\cdot), P, R\rangle$, where

- S is a finite set of the world states;
- S_0 is the initial state. The workflow of the service composition is executed from S_0.
- S_r represents a set of termination states. When a termination state is reached, the flow of a service composition is terminated.
- $A(.)$ is a finite set of the services, in which $A(s)$ represents the set of services that can be invoked in state $s \in S$;
- P is the probability distribution function. When a service a is invoked, the world makes a transition from its current state s to a succeeding state s'. The probability for this transition is labeled as $P(s'|s, a)$;
- R is the immediate reward function. When the current state is s, and a service a is invoked, we can get an immediate reward $R(s'|s, a, t)$, where t represents the time to call the service. In a service composition, the reward value is generally determined by the QoS attributes of the service.

For a service composition workflow, the $DWSC - MDP$ model can be used to describe it completely. Take the simple workflow in Fig. 6 as an example. This workflow can be described with a $DWSC - MDP$ model, in which the state set is $S = \{S_0, S_1, S_2, S_3, S_4\}$, the initial state is S_0, the termination state set is $\{S_4\}$. The services that can be invoked at different states include $A(S_0) = \{a_0, a_1\}$ and $A(S_1) = \{a_2, a_3, a_4, a_5\}$, etc. Examples of transition probability include $P(S_1 \mid S_0, a_0) = 1$, $P(S_2 \mid S_1, a_2) = 1$, etc. The reward value is calculated with the QoS obtained by invoking the service.

After the workflow is determined, service composition process starts from the initial state, selects a specific service for each state, and then moves to a new state. When a termination state is reached, the workflow of a service composition is completed. This workflow, which is composed of multiple specific services, is the result of a service composition. A good service composition result should be able to make the total reward value as large as possible.

4.3. Reward function in service composition

The reinforcement learning algorithm outputs the best action selection strategy using a Markov decision process model, making it suitable for the service selection problem. To use the reinforcement learning algorithm, we need to define the reward function first.

In service composition, the satisfaction level of invoking a service is usually determined by the QoS. Therefore, the reward function in this paper will be defined by the QoS attributes of a service. Since different QoS attributes may have different range of values, we first need to normalize different attributes and map them to [0, 1]. In addition, considering that some of the QoS attributes are positively correlated (such as throughput), and some of the QoS attributes are negatively correlated (such as response time), so the following two formulas are defined:

$$r = \frac{QoS - min}{max - min} \tag{15}$$

$$r = \frac{max - QoS}{max - min} \tag{16}$$

The formula (15) is used to normalize the QoS attributes that are positively correlated, and formula (16) is used to normalize the QoS attributes that are negatively correlated. r represents the result of normalization value of this attribute. QoS represents the QoS value of the attribute after the service is invoked, and max and min represent the maximum and minimum QoS values of the attribute.

If a service has multiple QoS attributes, the reward value will be calculated with formula (17).

$$R = \sum_{i=1}^{m} w_i * r_i \tag{17}$$

where R represents the total reward value, m is the number of QoS attributes considered, r_i represents the normalized value of the ith QoS attribute, and w_i represents the weight of the ith attribute. The weight reflects the importance of the different

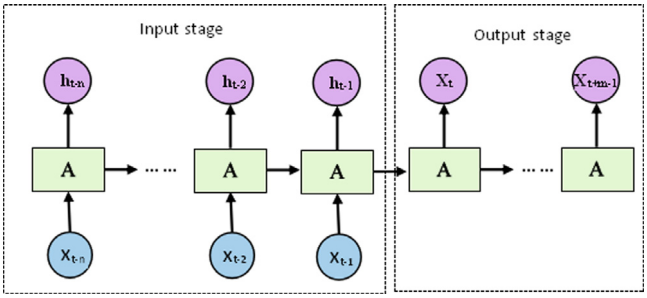


Fig. 9. The Structure of LSTM for Time Series Prediction.

attributes. Generally, it is set up according to the preference of the users on different attributes with $\sum_{i=1}^{m} w_i = 1$.

4.4. Time series prediction based on LSTM

In a dynamic network environment, the QoS often changes with time, and the reinforcement learning algorithm cannot estimate the changing trend of service quality. This section will introduce the method of time series prediction based on LSTM to help estimate the QoS in a dynamic environment.

The basic idea of the time series prediction is to predict the future QoS using the past data. This paper uses the LSTM neural network to predict the time series, which can be expressed as follows:

$$x_t = LSTM(x_{t-1}, x_{t-2}, x_{t-3}, \ldots, x_0; \theta) \tag{18}$$

where x_t represents the predicted data at the next time step, $LSTM(x_{t-1}, x_{t-2}, x_{t-3}, \ldots, x_0; \theta)$ represents the LSTM neural network with the parameter of θ. The input of the neural network is all past data. In practice, to predict with entire past data is often difficult to carry out. So the input of the LSTM neural network is actually the data of the previous n time steps. In addition, the LSTM neural network can predict more than one time steps. Therefore, the prediction of the LSTM neural network can be expressed in formula (19), in which m is the count of time steps of the prediction and the n is the count of time steps of the input.

$$x_{t+m-1}, \ldots, x_{t+1}, x_t = LSTM(x_{t-1}, x_{t-2}, \ldots, x_{t-n}; \theta) \tag{19}$$

After determining the input and output of the neural network, the LSTM neural network shown in Fig. 9 can be constructed. The graph shows the expansion of an LSTM neural network in time. That is, the neural network represented by A in the graph is the same neural network, and the data is passed from left to right in sequence of time. In the input phase, n time steps data are input to the LSTM neural network in turn. The output of the last time neural network will also be part of the current input. In the output phase, the LSTM neural network first outputs the prediction data x_t. After that, the neural network no longer has the input from the external data (because it is now the future time, which no longer has the actual data). It only takes the predicted value from the previous moment as the input of the current moment, as shown in Fig. 9, and outputs a predicted value in each step. When m predicted values are obtained, the prediction task completes.

After constructing LSTM neural network, a large amount of historical data is needed to train the neural network. The training data set needs to be segmented, and every continuous $n+m$ data points are used as the training data. The first n data is the input of the neural network, and the last m time steps data represents the expected prediction results to calculate the error of the neural network. The error is calculated by formula (20), in which x_t represents the actual value of the t moment, x'_t represents the predicted value at t moment, and N is the number of data points in the training set. The error function used is the mean square error:

$$L = \frac{1}{N} \sum_{t=0}^{N} (x'_t - x_t)^2 \tag{20}$$

After training, the gradient descent method in formula (9) is used to adjust the parameters of the neural network. The training process is repeated many times until the error value is stable at the minimum. After training, the neural network can be used to predict the future QoS.

4.5. Adaptive service composition algorithm

A new adaptive service combination algorithm is developed, which combines reinforcement learning and QoS prediction. Before introducing the algorithm, we need to solve two problems of QoS estimation in service composition.

First, in service composition, the quality of a candidate service is not only related to the QoS attribute of the service itself, but also the subsequent workflow resulting from the selection of this service. For example, although a service has better QoS attributes, the QoS in the subsequent process with selecting this service may be poor, resulting in poor overall service composition quality. Therefore, in the estimation of the QoS, we employ the Q-learning algorithm and use the Q value to represent the QoS in the service composition. According to the Q value formula (7), the Q value is actually a linear superposition of each service reward value, so it can also be estimated by the time series prediction method. In addition, if the Q value is estimated through the LSTM neural network, the formula (7) needs to be rewritten as follows:

$$Q_{real}(s, a) = R(s' \mid s, a) + \gamma \max LSTM(s', a') \tag{21}$$

where $LSTM(s', a')$ represents the predicted value of the LSTM neural network corresponding to the state s' and service a', which is the Q value estimation for the state–action pair (s', a').

In addition, with the running of the service composition algorithm, new data will continue to be added to the historical data set. Generally speaking, the neural network is trained in a fixed training set, and then applied to the new data. But in the process of service composition, historical data is gradually accumulated, so incremental neural network training is needed. A neural network training should be carried out if a certain amount of new data is added. Moreover, with the accumulation of historical data, these data may take up a lot of space and will also affect the efficiency of neural network training. Therefore, it is necessary to limit the size of the historical data set and discard earlier historical data, which can improve the training efficiency of the neural network. The detailed algorithm is given in Algorithm 2. We refer to the proposed algorithm as QP-RL (Q-Prediction Reinforcement Learning).

5. Experiments and analysis

In this section, we report the experimental results to verify the effectiveness and efficiency of QP-RL under a dynamic environment. The influence of different algorithm parameters will be analyzed as well.

5.1. Experiment setting

The experiments are conducted over commonly used WS-DREAM dataset [38], which describes real-world QoS evaluation

Algorithm 2: Service Composition Based on QoS Prediction and Reinforcement Learning

initialize $LSTM(s, a)$, random exploration parameters ϵ, input data length n, historical data capacity d, LSTM update threshold u;
input DWSC-MDP model;
repeat
  Set $s = s_0$;
  repeat
    Choose service a with $\epsilon - greedy$ strategy (Using LSTM to estimate the quality of service);
    Invoke a, get reward with formula (17), move to state s';
    Compute Q value with formula (21), add this data to historical data set;
    if data count in historical data set = d **then**
      Discarding the oldest historical data;
    end if
    if new data count in historical data set = u **then**
      Train LSTM with formula (20);
    end if
    $s \leftarrow s'$;
  until Achieve the termination state;
until Convergence (or up to the number of preset cycles);
Algorithm terminates. (The prediction Q value of each service can determine the best candidate service in each state to complete the service composition.)

Table 2
Samples of the WS-DREAM dataset.

Service	Response-time	Throughput
1	5.98	0.334
2	0.397	0.142
3	8	70.0
4	0.228	17.543
5	5.597	218.487
6	0.098	59.829

results from 339 users on 5,825 Web services. Some samples of QoS for a user to invoke different services are shown in Table 2. In our experiments, we use response-time and throughput to calculate the reward value of service invocation according to formulas (15), (16), (17). Other QoS attributes can be included in a similar way. We randomly generate the DWSC-MDP model with the structure shown in Fig. 10, where each hollow circle represents a state node. Each state node has two succeeding states that may be transferred to. Each transfer path has at least one possible candidate service. For example, state S_1 is transferred to state S_3 when the service a_0 or a_1 is invoked. It will be transferred to state S_4 when service a_3 or a_4 is invoked. The execution of the entire workflow requires M times of selection of candidate states, where each state has a number of candidate services. In particular, by setting the count of the service selection as m, the number of candidate services per state as n_a and the possible subsequent state number as n_s, we can randomly generate different service composition models.

The LSTM network in the algorithm is implemented using the Google deep learning framework TensorFlow [39], and the parameters of the neural network itself use the default settings. The experiments are conducted on 64 bits Ubuntu-16.04.04 with an i5-7500k CPU, 12G memory, and a nvdia-gtx1060 GPU.

5.2. Result analysis

5.2.1. Influence of parameters

In the QP-RL algorithm, there are some tunable parameters. This section will test and compare different parameters to find the best parameter settings. First, we randomly generate a DWSC-MDP model, where the count of service selection is set to 50,

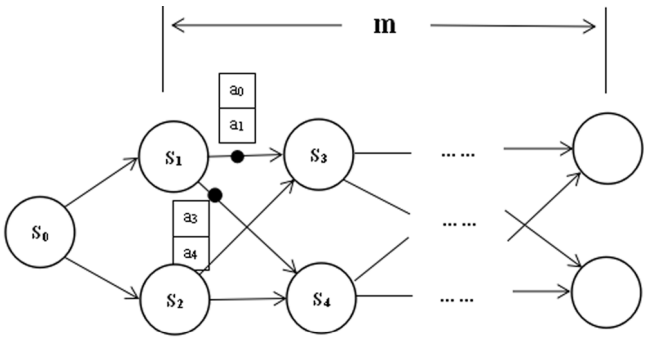


Fig. 10. Random DWSC-MDP model.

the number of candidate services per state is set to 4, and the possible subsequent state is set to 2. Second, in order to compare the values of different parameters, we set up a set of default parameters, in which the random exploration parameter ϵ is set to 0.6, the input data length n of the neural network is set to 20, the historical data set capacity d is set to 1000, the neural network update threshold u is set to 100. When comparing different values of a parameter, the other parameters will be set to the default values.

Figs. 11–14 show the results. There are two experimental results for each parameter. The first one is the line chart of the algorithm execution time with the parameter change, in which the longitudinal axis indicates the algorithm running time in hours. The second is the line chart of the cumulative reward value with the change of the parameter. The cumulative reward value is the sum of the reward obtained by each service invoked in the service composition. The greater the cumulative reward value, the better the result of the service composition.

For the random exploration parameter ϵ, as shown in Fig. 11, when the ϵ value is small, the algorithm is close to the random exploration. So the neural network is hard to be fully trained and the QoS cannot be accurately estimated. The inaccurate estimation of QoS results in smaller cumulative reward and longer running time. When ϵ is too large, the algorithm can get the final result quickly, but it is easy to fall into the local optimum. So the cumulative reward value is not the best. When ϵ is 0.6, the algorithm can get the best result of service composition and have a good efficiency, which is a better choice.

For the input data length n, it can be seen that the execution time of the algorithm increases sharply with the increase of n, so the length of the input data should not be set too large due to the hardware condition. In addition, although a larger n can get better results of the service composition, further improvement when n is more than 20 is not very obvious. But the execution time increases a lot. Thus, n is set to 20 to get a more balanced algorithm effect.

For historical dataset capacity d, from Fig. 13, it can be seen that when d is too small, the quality of the results of the service composition is very poor. This is because the neural network cannot be fully trained. With the increase of d, the effectiveness and execution time of the algorithm are improved. When d reaches 1000, the cumulative reward value tends to be stable, indicating that the oversized historical data capacity may not be of great help to the QoS estimation. As a result, d is set to 1000.

As shown in Fig. 14, the execution time of the algorithm decreases with the increase of u, but the improvement of the result is not obvious when u becomes too small. The reason is that the neural network is able to accurately predict the data for a period of time. So we do not need to update LSTM too frequently. According to the experimental results, u is set to 100 to minimize the execution time of the algorithm.

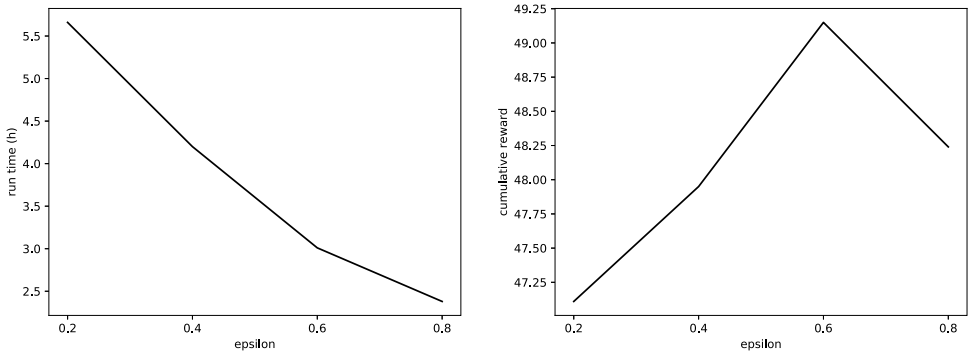


Fig. 11. Experiments for ϵ.

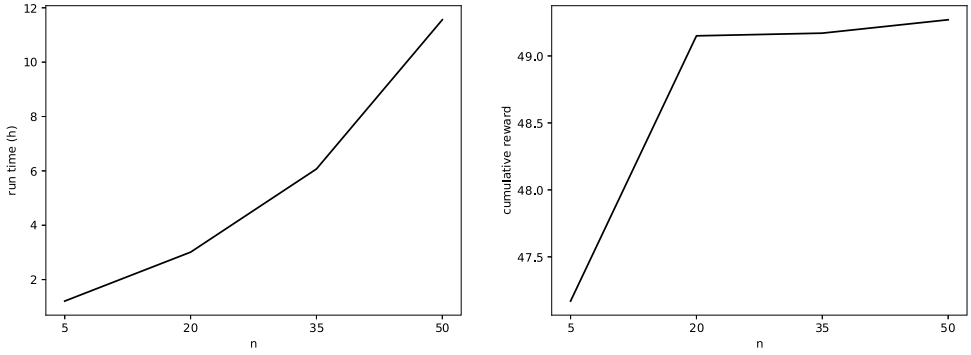


Fig. 12. Experiments for input data length n.

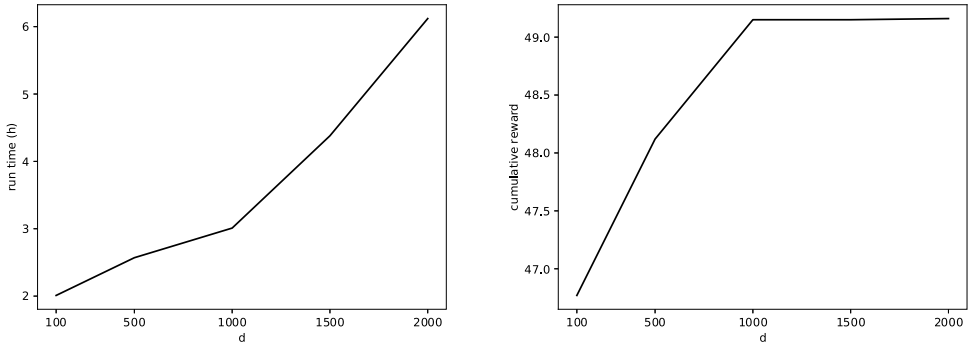


Fig. 13. Experiments for historical data capacity d.

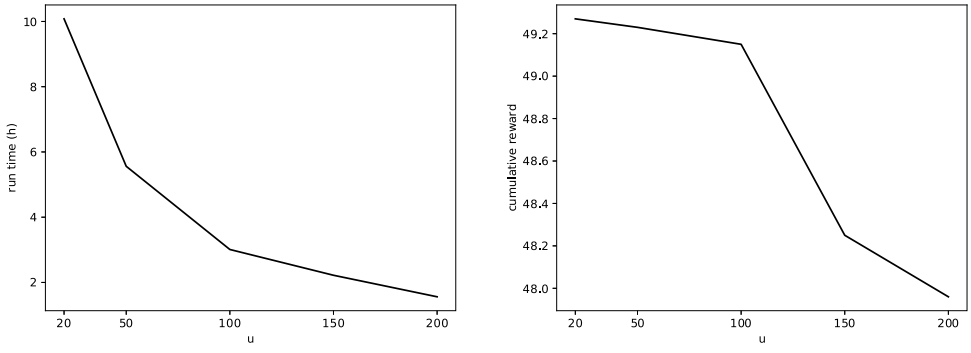


Fig. 14. Experiments for LSTM update threshold u.

5.2.2. The effectiveness of dynamic environment

The proposed service composition algorithm aims to deal with the changes of QoS in a dynamic environment. The QoS prediction technique is used to improve the accuracy of QoS estimation. In this section, we verify the effectiveness of the proposed method in a dynamic environment and compare it with the Q-learning

based adaptive service composition algorithm. In order to verify the performance of the algorithm in a highly dynamic environment, we have made some modifications to the original data, which is given by

$$x'_t = x_{t-1} + (1 + v)(x_t - x_{t-1}) \tag{22}$$

where x_t represents the predicted value of QoS, v represents the ratio of data changes and x'_t is new data. We control the degree of fluctuation of the data by changing the value of v. A larger the value of v leads to a greater the fluctuation degree of the data, which simulates the change of QoS in a highly dynamic environment.

Fig. 15 shows the result of Q-learning algorithm and QP-RL algorithm in the cases of v=0, v=0.1 and v=0.2. The horizontal axis is the number of episodes executed by the algorithm, and an episode is an execution process from the start state to the termination state. The vertical axis is the cumulative reward value of the result of the service composition. Through the comparison of the three graphs, it can be seen that the QP-RL algorithm can eventually achieve a more stable state, while the Q-learning algorithm is becoming more and more unstable with the increase of the degree of data fluctuation. The reason for this is that the QP-RL algorithm uses historical data to predict the time series of the quality of service. Although the volatility of the data has become larger, recurrent neural network is still able to predict these changes very well. The Q-learning algorithm estimates the quality of service based on the Q value table. When the data fluctuation becomes larger, the error of the estimation of QoS will become larger, which will result in the deterioration of the quality of the service composition and the instability of the algorithm.

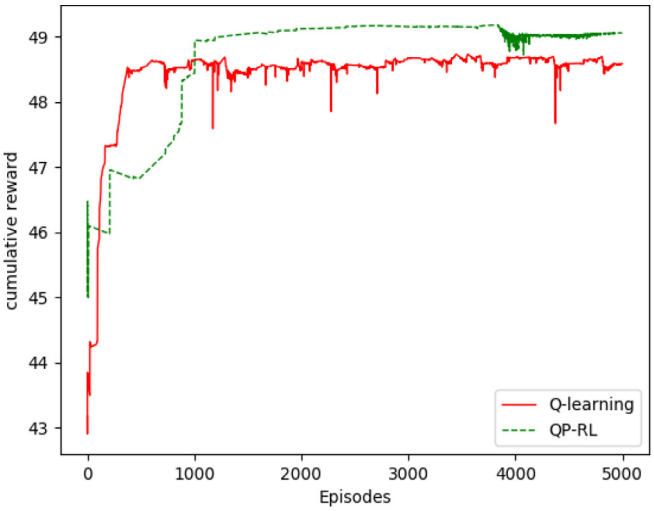
In addition, we can see from the comparison between Q-learning and QP-RL algorithm that the result of Q-learning algorithm is superior to QP-RL algorithm in the initial stage of algorithm execution. This is because in the early stage of execution of the algorithm, the data in the neural network historical data set is insufficient. So the prediction effect and the quality of the service composition is very poor. However, with the running of the algorithm, the neural network finally has stronger prediction ability for the dynamic data. Compared with Q-learning, it can predict the change of data more accurately, and the stability of the algorithm is much better than the Q-learning based service composition algorithm. After the convergence of the Q-learning algorithm, the result of the service composition still has a great fluctuation. The reason is that Q-learning cannot make a good estimate of the future data changes, resulting in the unstable results of the final service composition.

The experimental results show that the proposed QP-RL algorithm can get better service composition results in dynamic environment and achieve better adaptive ability and stability.

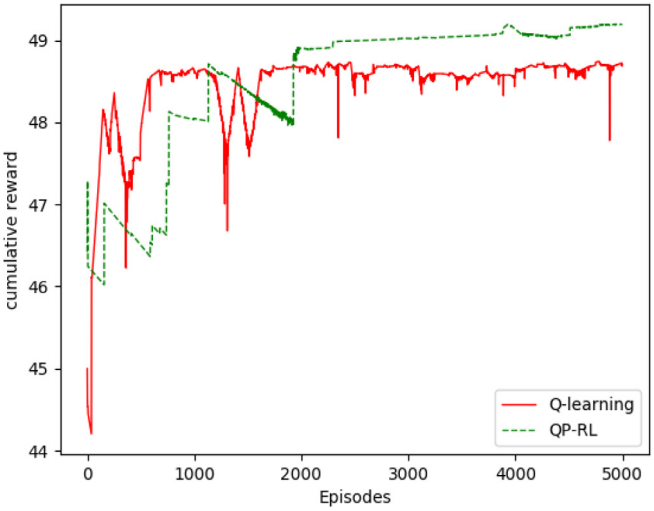
5.2.3. The effectiveness under the different scale problems

The last section verifies the effectiveness of QP-RL algorithm in a dynamic environment. This section will test the effectiveness and efficiency of the algorithm. In this experiment, three DWSC-MDP models are randomly generated to simulate different scales of service composition problems. The number of service selection, number of candidate services per state, and number of possible subsequent states of the three models are set to (20,2,2), (50,4,2), (100,10,5) respectively, corresponding to small, medium, and large-scale problems.

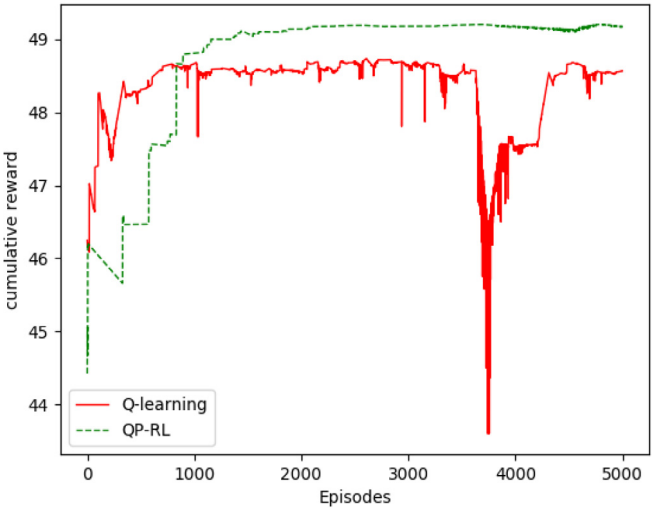
Fig. 16 shows the running process of Q-learning algorithm and QP-RL algorithm under three cases. It can be seen that the running process of the algorithm is similar to that in the last section. The results of QP-RL algorithm at the initial stage are poor in quality. As the algorithm continues to run, the result quality of the QP-RL algorithm is getting better and better. Finally, it is more



(a) v=0

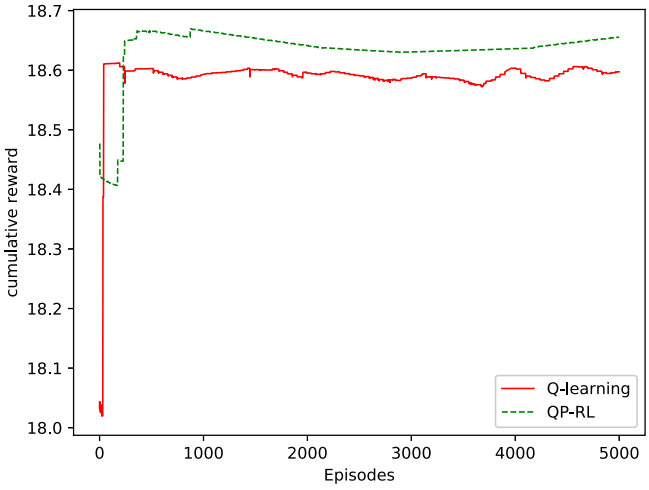


(b) v=0.1

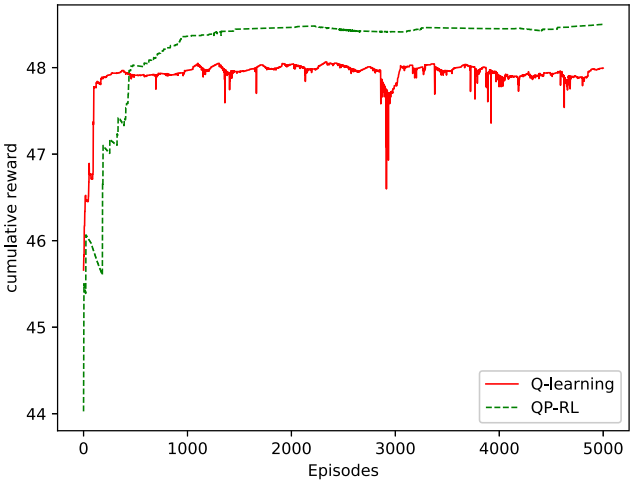


(c) v=0.2

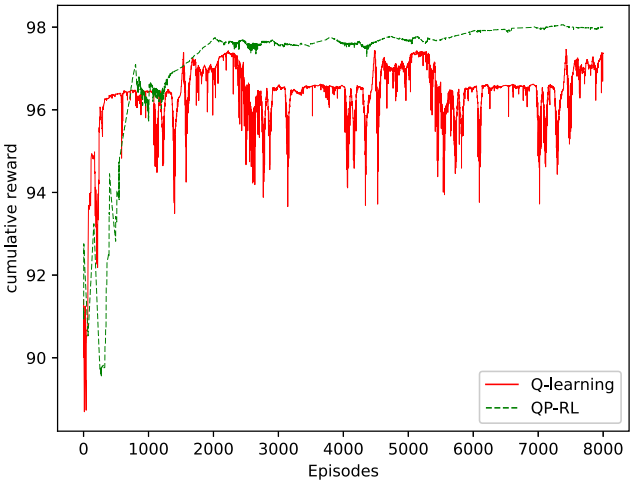
Fig. 15. Experiments for different v.



(a) small-scale



(b) medium-scale



(c) large-scale

Fig. 16. Experiments under different scale problems.

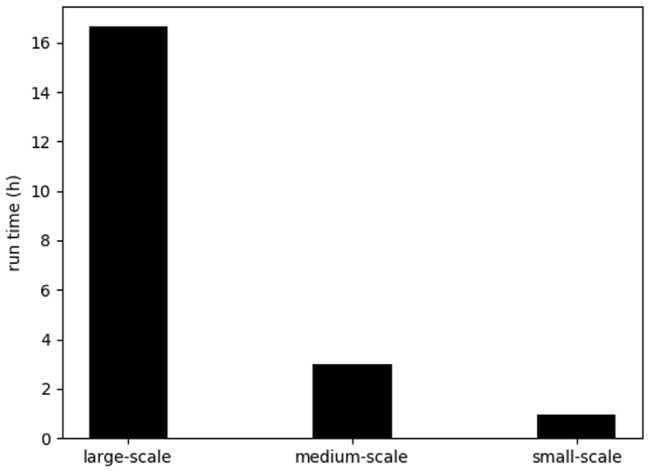


Fig. 17. Execution time under different scale problems.

stable than the result of the service composition obtained by the Q-learning algorithm.

The above results show that in large-scale problems, the QP-RL algorithm can still get better service composition results. It is also observed that when the scale of problem is large, the execution time of the QP-RL algorithm will be significantly longer. Fig. 17 shows the execution time of QP-RL algorithm under different scales. It can be seen that as the scale of the problem increases, the execution time of the QP-RL algorithm will also increase dramatically. For an extremely large-scale problem, the QP-RL algorithm may fail to complete. For most practical service composition scenarios, QP-RL algorithm can be effectively applied. What is more, the time consumption of the QP-RL algorithm is mainly in the training of a large number of neural networks. But for the users, it is only necessary to use the trained neural network to determine the candidate services for the service composition. Therefore, the neural network training and learning process in the QP-RL algorithm can be performed using a high-performance computers, and only the final result of the service composition is provided to the users.

5.3. Remarks

In this section, we verify the effectiveness of the QP-RL algorithm under a dynamic environment by a series of experiments, and compare it with the Q-learning based service composition algorithm. Different parameters of the QP-RL algorithm are tested and the optimum selection of the parameters is discussed. The experimental results show that the adaptive service composition algorithm that combines QoS prediction and reinforcement learning can be well applied to dynamic changing environment. The result of service composition is superior to the Q-learning algorithm, and has good stability.

In addition to the above advantages, the experiments also show that the algorithm proposed in this paper has some shortcomings. The QP-RL algorithm has poor quality of service composition in the early stage. For this problem, we can pre-train the neural network with historical data. This may help achieve a better service composition results from the beginning. In addition, when dealing with large-scale problems, the proposed method can achieve better service combination results, but the efficiency of the algorithm becomes slow. We consider to employ the multi-agent technology to achieve better scalability as part of our future work.

6. Conclusion and future work

Service composition offers one of the most important ways of software reuse by combining existing web services to form new systems that meet the complex requirements of users. With the increase of functionally similar services, there is a need to select appropriate services from a large candidate pool to form a high-quality composite service. However, in a dynamic network environment, the QoS is not static, and the quality may fluctuate with time. So the service composition method needs to be adjusted dynamically to adapt to the changes of the environment. The proposed approach takes into account the characteristics of the dynamic change of QoS. A recurrent neural network based model is developed to estimate the QoS time series. A DWSC-MDP model is used to formally represent a service composition. An adaptive service composition algorithm is developed that integrates reinforcement learning and dynamic QoS prediction. Experimental results and comparison with the Q-learning based adaptive service composition algorithm help demonstrate the effectiveness of the proposed approach.

The proposed method of adaptive service composition based on reinforcement learning uses a Q value table and recurrent neural networks. This requires the establishment of a corresponding recurrent neural network for every possible candidate service, which leads to a high computational cost for a large number of candidate services. For this problem, we plan to explore deep reinforcement learning and use neural network mapping to replace the Q-value table in our future work. We also plan to explore a multi-agent system where multiple agents communicate and coordinate with each other to improve the efficiency.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by National Key Research and Development Plan, China (No. 2018YFB1003800) and NSFC Projects (Nos. 61672152, 61532013), Collaborative Innovation Centers of Novel Software Technology and Industrialization and Wireless Communications Technology, China. Qi Yu's work was sponsored in part by US National Science Foundation under grant IIS-1814450. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agency.

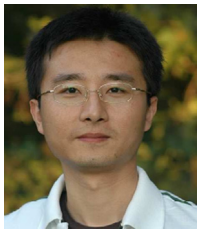
References

[1] K. Sangsanit, W. Kurutach, S. Phoomvuthisarn, REST web service composition: A survey of automation and techniques, in: 2018 International Conference on Information Networking, ICOIN, 2018, pp. 116–121.
[2] Y. Yan, P. Poizat, L. Zhao, Repair vs. recomposition for broken service compositions, in: International Conference on Service-Oriented Computing, Springer, 2010, pp. 152–166.
[3] D. Ardagna, B. Pernici, Adaptive service composition in flexible processes, IEEE Trans. Softw. Eng. 33 (6) (2007) 369–384.
[4] F. Li, L. Zhang, Y. Liu, Y. Laili, Qos-aware service composition in cloud manufacturing: a Gale-Shapley algorithm-based approach, IEEE Trans. Syst. Man Cybern. Syst. (2018) 1–12.
[5] Y. Yan, P. Poizat, L. Zhao, Self-adaptive service composition through graph-plan repair, in: Web Services (ICWS), 2010 IEEE International Conference on, IEEE, 2010, pp. 624–627.
[6] Y. Yan, P. Poizat, L. Zhao, Repairing service compositions in a changing world, in: Software Engineering Research, Management and Applications 2010, Springer, 2010, pp. 17–36.
[7] M. Alrifai, D. Skoutas, T. Risse, Selecting skyline services for qos-based web service composition, in: Proceedings of the 19th International Conference on World Wide Web, ACM, 2010, pp. 11–20.
[8] V.A. Permadi, B.J. Santoso, Efficient skyline-based web service composition with QoS-awareness and budget constraint, in: 2018 International Conference on Information and Communications Technology, ICOIACT, 2018, pp. 855–860.
[9] O. Hammas, S.B. Yahia, S.B. Ahmed, Adaptive web service composition insuring global QoS optimization, in: 2015 International Symposium on Networks, Computers and Communications, ISNCC, IEEE, 2015, pp. 1–6.
[10] S.K. Gavvala, C. Jatoth, G. Gangadharan, R. Buyya, Qos-aware cloud service composition using eagle strategy, Future Gener. Comput. Syst. 90 (2019) 273–290.
[11] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, A. Bouguettaya, Adaptive service composition based on reinforcement learning, in: International Conference on Service-Oriented Computing, Springer, 2010, pp. 92–107.
[12] H. Wang, G. Huang, Q. Yu, Automatic hierarchical reinforcement learning for efficient large-scale service composition, in: 2016 IEEE International Conference on Web Services, ICWS, 2016, pp. 57–64.
[13] H. Wang, X. Chen, Q. Wu, Q. Yu, Z. Zheng, A. Bouguettaya, Integrating on-policy reinforcement learning with multi-agent techniques for adaptive service composition, in: Service-Oriented Computing, Springer, 2014, pp. 154–168.
[14] y. lei, P.S. Yu, Multi-agent reinforcement learning for service composition, in: 2016 IEEE International Conference on Services Computing, SCC, 2016, pp. 790–793.
[15] P. Kendrick, T. Baker, Z. Maamar, A. Hussain, R. Buyya, D. Al-Jumeily, An efficient multi-cloud service composition using a distributed multiagent-based, memory-driven approach, IEEE Trans. Sustain. Comput. (2019) 1–13.
[16] W. Li, J. Cao, K. Hu, J. Xu, R. Buyya, A trust-based agent learning model for service composition in mobile cloud computing environments, IEEE Access 7 (2019) 34207–34226.
[17] D. Billsus, M.J. Pazzani, Learning collaborative information filters, in: Icml, vol. 98, 1998, pp. 46–54.
[18] J.L. Herlocker, J.A. Konstan, A. Borchers, J. Riedl, An algorithmic framework for performing collaborative filtering, in: SIGIR '99: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15–19, 1999, Berkeley, CA, USA, 1999, pp. 230–237.
[19] H.N. Kim, A.T. Ji, I. Ha, G.S. Jo, Collaborative filtering based on collaborative tagging for enhancing the quality of recommendation, Electron. Commer. Res. Appl. 9 (1) (2010) 73–83.
[20] K. Chen, H. Mao, X. Shi, Y. Xu, A. Liu, Trust-aware and location-based collaborative filtering for web service QoS prediction, in: 2017 IEEE 41st Annual Computer Software and Applications Conference, COMPSAC, vol. 2, 2017, pp. 143–148.
[21] C. Wu, W. Qiu, X. Wang, Z. Zheng, X. Yang, Time-aware and sparsity-tolerant qos prediction based on collaborative filtering, in: 2016 IEEE International Conference on Web Services, ICWS, 2016, pp. 637–640.
[22] G. Zou, M. Jiang, S. Niu, H. Wu, S. Pang, Y. Gan, Qos-aware web service recommendation with reinforced collaborative filtering, in: International Conference on Service-Oriented Computing, Springer, 2018, pp. 430–445.
[23] R.N. Calheiros, E. Masoumi, R. Ranjan, R. Buyya, Workload prediction using ARIMA model and its impact on cloud applications x2019; QoS, IEEE Trans. Cloud Comput. 3 (4) (2015) 449–458.
[24] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 3104–3112.
[25] A. Graves, A.-r. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: Acoustics, Speech and Signal Processing (Icassp), 2013 Ieee International Conference on, IEEE, 2013, pp. 6645–6649.
[26] R.-G. Cirstea, D.-V. Micu, G.-M. Muresan, C. Guo, B. Yang, Correlated time series forecasting using multi-task deep neural networks, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, ACM, 2018, pp. 1527–1530.
[27] L. Yunpeng, H. Di, B. Junpeng, Q. Yong, Multi-step ahead time series forecasting for different data patterns based on LSTM recurrent neural network, in: 2017 14th Web Information Systems and Applications Conference, WISA, 2017, pp. 305–310.
[28] L.P. Kaelbling, A situated-automata approach to the design of embedded agents, ACM SIGART Bull. 2 (4) (1991) 85–88.
[29] D.E. Goldberg, J.H. Holland, Genetic algorithms and machine learning, Mach. Learn. 3 (2) (1988) 95–99.
[30] R.H. Crites, A.G. Barto, Elevator group control using multiple reinforcement learning agents, Mach. Learn. 33 (2–3) (1998) 235–262.
[31] J.D. Hamilton, Time Series Analysis, Princeton University Press, Princeton, NJ, 1994.
[32] J. Contreras, R. Espinola, F.J. Nogales, A.J. Conejo, ARIMA Models to predict next-day electricity prices, IEEE Trans. Power Syst. 18 (3) (2003) 1014–1020.

[33] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[34] J.T. Connor, R.D. Martin, L.E. Atlas, Recurrent neural networks and robust time series prediction, IEEE Trans. Neural Netw. 5 (2) (1994) 240–254.
[35] A. Karpathy, J. Johnson, L. Fei-Fei, Visualizing and understanding recurrent networks, 2015, ArXiv preprint, arXiv:1506.02078.
[36] C. Goller, A. Kuchler, Learning task-dependent distributed representations by backpropagation through structure, in: Neural Networks, 1996, IEEE International Conference on, vol. 1, IEEE, 1996, pp. 347–352.
[37] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[38] R. Zhu, D. Niu, Z. Li, Robust web service recommendation via quantile matrix factorization, in: INFOCOM 2017 - IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9.
[39] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, Tensorflow: large-scale machine learning on heterogeneous distributed systems, 2016, Software available from tensorflow.org.

Hongbing Wang is a professor from School of Computer Science and Engineering, Southeast University, China. He received his Ph.D. in computer science from Nanjing University, China. His research interests include Service Computing, Cloud Computing, and Software Engineering. He published more than fifty refereed papers in international conferences and Journals, e.g., Journal of Web Semantics, JSS, TSC, ICSOC, ICWS, SCC, CIKM, ICTAI, WI, etc. He is a member of the IEEE.

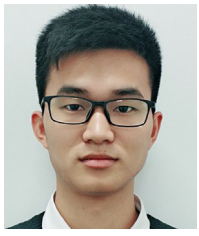
Jiajie Li is a postgraduate student (2015–2018) in the School of Computer Science and Engineering, Southeast University, PR China. His research focuses on service selection and service composition.



Qi Yu received the Ph.D. degree in computer science from Virginia Polytechnic Institute and State University (Virginia Tech). He is an associate professor in the College of Computing and Information Sciences at the Rochester Institute of Technology (RIT). His research interests lie in the areas of machine learning, data mining, and service computing. He has published over 80 papers in these fields. He is a member of the IEEE.

Tianjing Hong is a postgraduate student in the School of Computer Science and Engineering, Southeast University, PR China. Her research focuses on service selection and composition and service prediction.

Jia Yan is a postgraduate student in the School of Computer Science and Engineering, Southeast University, PR China. His research focuses on service selection and service composition.



Wei Zhao is a postgraduate student in the School of Computer Science and Engineering, Southeast University, PR China. His research focuses on service selection and service composition.