International Journal of Innovative Computing, Information and Control
Volume **9**, Number **6**, June **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 2361–2376

QOS OPTIMIZATION FOR WEB SERVICES COMPOSITION BASED ON REINFORCEMENT LEARNING

Liang-Bing Feng[1,2], Masanao Obayashi[1], Takashi Kuremoto[1]
Kunikazu Kobayashi[3] and Shun Watanabe[1]

[1]Graduate School of Science and Engineering
Yamaguchi University
2-16-1 Tokiwadai, Ube, Yamaguchi 755-8611, Japan
{ m.obayas; wu; s005we }@yamaguchi-u.ac.jp

[2]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
No. 1068, Xueyuan Avenue, Shenzhen University Town, Shenzhen 518055, P. R. China
fengliangbing@gmail.com

[3]School of Information Science and Technology
Aichi Prefectural University
Ibaragabasama 1522-3, Nagakute, Aichi 480-1198, Japan
kobayashi@ist.aichi-pu.ac.jp

Received March 2012; revised September 2012

Abstract. *To get an efficient Web services composition, QoS (Quality of Service) of composite service needs to be optimized during the dynamical generation of service flow. However, current Web services composition solutions are often not as real-time and adaptable as expected because dynamic obtaining and utilizing of QoS are not entirely realized. In this paper, a novel services composition framework is proposed. Three mainly works are done in this framework. 1) For QoS optimizing, an improved Hierarchical Reinforcement Learning method (HRL) is used to deal with the hierarchical architecture of Web services composition. 2) An extensible QoS model, the QoS value of which is obtained through learning process in the dynamical service execution, is founded. 3) Instead of pre-defined composite service in conventional methods, a Web service dynamical composing approach which is driven by business flow rules is proposed for real-time optimizing QoS. Finally, to confirm the effectiveness of our proposed method, a simulation of Web services composition is executed and its QoS is optimized by the proposed method.*
Keywords: Web services composition, Quality of Service (QoS), Reinforcement learning

1. **Introduction.** Web service is component-based software entity which loosely couples and encapsulates discrete functions. Web services composition is a special composite Web service which is logically composed by Web services. Here, Web services invoke each other and generate composite service for offering a service function to users. The composite service performance can be evaluated with two aspects. One is functional evaluation. In this aspect, the service providing function should be completely matched with user's request. The other is non-functional evaluation – Quality of Service (QoS). To satisfy users mostly, the QoS of composite services needs to be optimized during service flow dynamic generation. Actually, QoS is variable in the dynamic environment. With the increasing number of Web services being made available on the Internet, it is difficult to optimize QoS during the dynamic generation of services flow. To design a Web services composition system that can leverage, aggregate and make use of individual component's QoS information to derive the optimal QoS of the composite service is still an ongoing research problem [1]. To real-timely obtain and utilize QoS, we proposed a novel intelligent

agent which can dynamically evaluate QoS, choose services, composite service. There are three key aspects of our work:

• An extensible QoS model for dynamic obtaining is proposed. The characteristic of QoS can be updated in time through monitoring the execution of service.

• An HRL method is adapted to QoS optimization. The hierarchical reward obtaining approach is offered by HRL according to hierarchical architecture of Web services composition.

• A Web services composition framework in which service flow is generated by rule constrains is proposed. In this framework, the service which has optimal QoS can freely be selected and composed while the services composition is executed. Therefore, it is realized that the services functional composition and QoS optimization combine together.

The rest of this paper is organized as follows. Section 2 introduces some related works. Section 3 states a novel QoS model. In Section 4, we describe a Web services composition framework in which services flow is generated by rule constraints. Section 5 illustrates a method of execution time optimization for Web services composition. In Section 6, we describe the multi-dimensional QoS optimization. Section 7 presents experimental results of using the proposed method. Finally, Section 8 summarizes the paper and discusses some directions for the future work.

2. **Overview of Related Works.** While this paper is mainly concerned with QoS optimization of Web services composition, we provide a concise review on it.

2.1. **QoS optimization approaches for Web services composition.** Because QoS is a major concern in the design and management of a composite service, QoS optimization algorithm has become one of focuses for Web services composition. Paper [3] presents a middleware platform which addresses an integer programming method of QoS optimization. Two selection approaches are described and compared: one based on local (task-level) selection of services, and the other based on global allocation of tasks to services. According to [10], the two different configuration methods for Web services composition are used. The reliability of Web services composition is optimized. For dynamically selecting services, Reinforcement Learning (RL) selection methods related to Web services can be found in several papers. Paper [5] focuses on the following issues: Web services composition models, such as Petri net or hype-graph, are established; the QoS of every transition (task) is learned; the exhaustive search method is used for searching the Web services composition model and calculating the QoS of every path; find the path according to the user's requests. According to [11], the degree for exploration and exploitation of RL has been studied for obtaining appropriate exploration which suits to learn QoS of Web services. In these papers, the optimal service is selected by exhaustively searching. When composition steps and numbers of the same function services increase, the services composition states increase exponentially. Paper [7] mentioned an integration service selection of function to reduce the scale of problem, but did not show how to realize the integration. As the Web services functions are composed hierarchically, here we consider that QoS can be optimized hierarchically to decrease the number of composition states.

2.2. **QoS obtaining method.** The QoS models have widely been studied [3,6,7]. Liu et al. classify the QoS with generic quality criteria and business related criteria in [7]. The value of QoS is represented as matrix and is able to be set by users. There is a QoS registry to store every service's QoS value. The user can read these QoS value for selecting services. In [3], QoS is modeled with execution price, execution duration, reputation, reliability, availability. In most of the current approaches, QoS information acquisition relies on services providers to advertise their QoS information or provide an interface to access

the QoS values, which is subject to manipulation of the providers. Obviously, services providers may not advertise their QoS information in a "neutral" manner, for example, execution duration, reliability [7]. QoS of Web service is dynamic, such as execution time which is decided by real-time execution environment. We support that the objective QoS value should be collected through active monitoring while a Web service is executed.

2.3. **Dynamic composition of Web services.** To optimize QoS for Web services composition, the QoS of every elementary service needs to be learned on line and the services composition flow needs to be dynamic changed for freely choosing services. The flow of the services composition needs to be dynamically generated. Because business service is expressed as web service, the services and business flows change frequently in the actual environment. This needs that services composition dynamically select services. However, current Web services composition development and management is mostly a manual activity that requires specific knowledge [8]. This may be applied to available standards, such as BPEL4WS (Business Process Execution Language for Web Services). Consequently, the services flow is pre-defined and difficultly evolves when it is executed. To realize QoS optimization, the efficient method for services flow dynamic generation is expected. According to [8], the dynamic services composition is controlled and managed by a rule which is abstracted from the Web services flow. A Colombo model is created in [9]. The service behaviors are recorded by "World State". The composition is implemented by atomic operation changing the "World State". The atomic operation is trigged by Proposition Dynamic Logic (PDL). However, these methods did not realize free services selection and compositing during service execution. These papers have discussed some methods of services flow dynamical generation. However, the work of the optimal QoS services selection for Web services composition is limited in these papers.

3. **The QoS Model for Web Service.** Actually, with the ever increasing number of same function Web services being made available on the Internet, there is a need to distinguish them using a set of well-defined QoS criteria [3]. This is considered as the basic principle of QoS optimization. The pre-researches have proposed many quality criteria for measuring QoS of elementary services. It includes execution price, execution duration, security, reputation, reliability, availability, and so on [3,7]. In this section, we study the characteristic of these quality criteria and separate these into three kinds of qualities: execution time, execution cost and usability.

Execution time: The execution time (response time) of an elementary service has been studied [12]. The service execution time is dynamic and its distribution obeys an exponential distribution. The distribution function of execution time $T_{service}$ is expressed as:

$$F(T_{service};\lambda) = \begin{cases} 1 - e^{-\lambda T_{service}}, & T_{service} > 0 \\ 0, & T_{service} \leq 0. \end{cases} \tag{1}$$

In this function, λ is the execution rate of service.

Execution cost: The execution cost is the fee or price of an invoked service. Web service represents the business service, and execution price is different at the different time according to the business. For example, a hotel subscribing service may have several prices when it is used at different seasons. We express the execution cost as below function, here, $C_{service}$ is the execute cost of a service.

$$C_{service} = \begin{cases} c_1 \text{ at probability } p_1 \\ c_2 \text{ at probability } p_2 \\ \cdots\cdots\cdots \\ c_n \text{ at probability } p_n \end{cases} \tag{2}$$

where $p_1 + p_2 + \ldots + p_n = 1$.

Usability: The usability is a measure whether a service is correctly used [13]. This property includes reliability, availability, security, reputation, and so on. The probability that a request is correctly responded within maximum expected time is called reliability Q_r. The probability that the service is accessible is called availability Q_a. Security Q_s is the probability of service meeting the user's security request. Reputation Q_{re} is a measure of service trustworthiness set by user. Furthermore, the usability can also be extended to other property of service which relates to whether the service being correctly used, such as integrity, interoperability, robustness. The probability of usability $U_{service}$ can be calculated using probability addition function from $P(Q_r)$, $P(Q_a)$, $P(Q_s)$, $P(Q_{re})$. When reliability, availability, security, and so on are monitored during the service execution, the usability can be calculated by Formula (3); here, $U_{service}$ is the usability of a service.

$$\begin{aligned} U_{service} = {} & P(Q_r) + P(Q_a) + P(Q_s) + P(Q_{re}) - P(Q_r)P(Q_{re}) \\ & - P(Q_r)P(Q_s) - P(Q_a)P(Q_{re}) - P(Q_r)P(Q_a) \\ & - P(Q_a)P(Q_s) - P(Q_s)P(Q_{re}) \\ & + P(Q_r)P(Q_a)P(Q_s) + P(Q_r)P(Q_a)P(Q_{re}) \\ & + P(Q_r)P(Q_s)P(Q_{re}) + P(Q_a)P(Q_s)P(Q_{re}) \\ & + P(Q_r)P(Q_a)P(Q_s)P(Q_{re}) \end{aligned} \tag{3}$$

The vector of QoS is defined by the following expression:

$$Q(_{service}) = (T_{service}, C_{service}, U_{service}). \tag{4}$$

The method for computing the value of the quality criteria is not unique. Other computation methods can be designed to fit the needs of specific applications [7].

4. Web Services Dynamic Composition and Its Petri Net Model.

In the process of Web service composition, there may be several services which have the same function and QoS of these services are dynamic. One service which has the best QoS needs to be selected. So, Web services composition flow cannot be pre-defined. Services need to be dynamically selected and composed. In order to realize dynamic services composing and services flow dynamic evolving, the business logics are translated into services composition rules. The services composition flow is generated by services composition rules. To formalize Web services composition flow, we introduce Petri net to model Web services composition.

4.1. The Petri net model for Web services composition.

As a graphical and mathematical tool, Petri net can intuitionally describe the flow logic of Web service [14]. We use stochastic high level Petri net (SHLPN) to visually describe services composition and represent some of services dependencies. A Web service task is represented as a transition of Petri net. The invoking relation is represented as the directional arc. Here, Tokens are colored and the color of Token represents data of service operation. It is available to change the color of Token by transition according to arc weight function. The execution ability of Web service is expressed by a transition velocity λ. The arc which connects place to transition becomes a forbidden arc when the transition is unusable. Sequential, cyclic, alternative and parallel compositions are four typical compositions of Web service. Here, the four basic types of Web services compositions are modeled as shown in figures from Figure 1, where $\bigcirc$, $\square$, $\rightarrow$ and $\bullet$ represent place, transition, arc and Token, respectively [14].

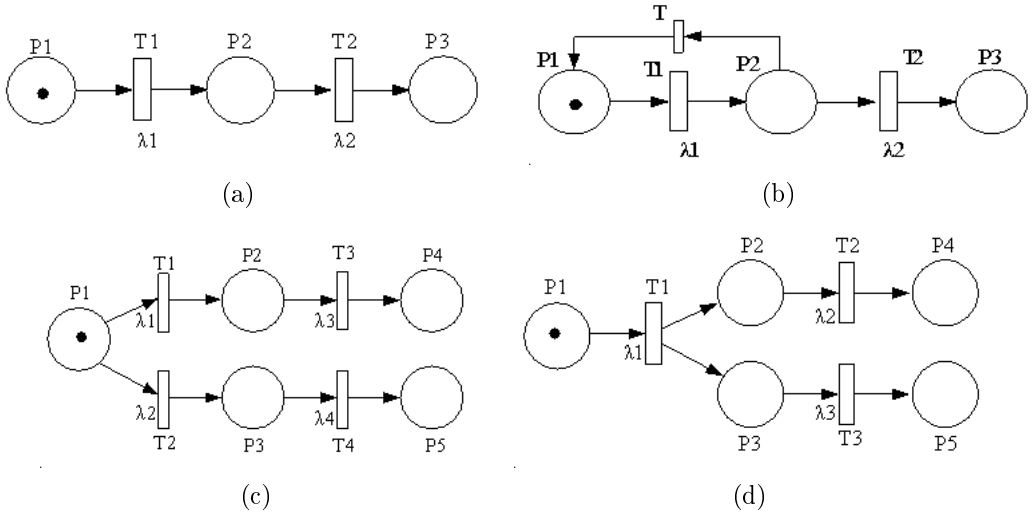


FIGURE 1. Petri net model for Web services composition: (a) sequential; (b) cyclic; (c) alternative; (d) parallel

4.2. **Web services flow generation by rule driven.** Web services composition expressed a business process and business process has its logic. The business flow can be abstracted and turned into composition rules. These rules drive the composition. The advantage of this framework is that it makes the definition, scheduling, construction and execution of services composition flexible and controllable, since composition elements can be combined in a plug and play manner by using composition rules. This makes it possible to generate a services composition when services are being executed. The changes of flow logic can be easily incorporated and effectuated through the redefinition of the appropriate composition rules [19]. Services which have different QoS can be freely selected and executed. This is different from pre-research on Web services flow generation. Conventionally, each elementary service is concrete service in Web services flow which is defined by BPEL4WS and cannot be changed when services composition is executing. The new services flow generation method is elaborated in the following.

1) The composition Flow Rule (FR). The business flow logic is translated into services composition rules. For example, travel booking services for a travel agent may include several sequential steps, such as accepting request, travel plan, several subscribing, insurance buying and paying. The subscribing service may have several parallel operations such as car subscribing, scenic-spot subscribing, hotel subscribing and itself is iterative for subscribing several different scenic-spots. The insurances service may also include several kinds of the same function services for user choosing. Suppose there are five types' insurance services. The paying service used to be several alterative operations such as check paying, credit card or cash paying. We can abstract composition rules of the travel booking service and turn them into Flow Rule as below. Here, $\odot$, $\oplus$, $\Diamond$, μ represent sequence, parallel, alternative, iteration composition respectively [15].

$$\text{FR1: Travel\_booking} = \text{Booking\_accept} \odot \text{Travel\_plan} \odot \text{Subscribe} \odot \text{Insurance} \odot \text{Pay} \quad (5)$$

$$\text{FR2: Subscribe} = \text{Scenic-spot\_booking} \oplus \text{Car\_booking} \oplus \text{Hotel\_booking} \quad (6)$$

$$\text{FR3: Subscribe} = \mu\text{Subscribe} \quad (7)$$

$$\text{FR4: Insurance} = \text{Insurance1} \Diamond \text{Insurance2} \Diamond \text{Insurance3} \Diamond \text{Insurance4} \Diamond \text{Insurance5} \quad (8)$$

$$\text{FR5: Pay} = \text{Pay\_cash} \Diamond \text{Pay\_check} \Diamond \text{Pay\_credit} \quad (9)$$

2) Web services reservoir. In rules of services composition, the services are abstract services S = $\{S_1, S_2, \ldots, S_n\}$. An abstract service is a non-executable service describing the core functionality in terms of operations, input, output, pre- and post-conditions. For each abstract service S_p, a set of concrete services $\langle S_{p1}, S_{p2}, \ldots, S_{pm}\rangle$ is available that implements the corresponding function of abstract service. Here, the concrete S_{pi} has a Web service's Internet site. When a services composition needs the function of service S_p, a service in $S_{p1}, S_{p2}, \ldots, S_{pm}$ is invoked and accomplishes the service function. This reservoir can be constituted in a local UDDI.

4.3. **The architecture of composition middleware.** A middleware is constituted to realize dynamic generation of the Web services composition flow and optimizing the QoS of composite service. Architecture of the middleware is proposed in Figure 2. When the middleware receives a user invocation, this invocation is transferred as a set of abstract services according to flow rule. The concrete elementary services are selected using a selection algorithm to accomplish abstract services' function. These concrete elementary services are found in services reservoir and their invocation messages are sent to the required IP addresses. When these elementary services are executed, their QoS and other characteristics are monitored and used to update the middleware. QoS optimization method used in the middleware will be stated in Sections 5 and 6.

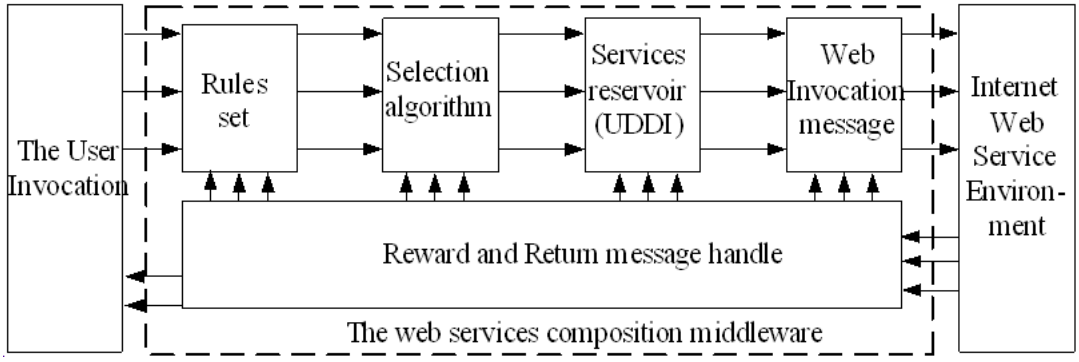


FIGURE 2. Middleware architecture for Web services composition

When a new business process is developed, the process flow is transformed into composition rules and stored in the middleware. When a user sends an invocation to the middleware, it is hierarchically decomposed by the rules. Then, the services composition component, which may be elementary Web service or composite service, is selected and hierarchically composed. Here, we compose the travel booking service according to Formulae (5)-(9) and list the services composition Petri model in Figure 3. When the middleware receives the user's request, for example, Travel_booking shown in Figure 3, it is decomposed into multiple sub-services according to FR1. The sub-services Book_accept, Travel_plan, Subscribe, Insurance and Pay, are decomposed again into their sub-services according to FR2-FR5 until every sub-services is an elementary service. The transition of Petri net also can be looked as a sub-Petri net for its function may be realized by a composite service [15]. For example, the subscribing sub-service is looked as a transition in Figure 3. This transition is a sub-Petri net which consists of several transitions and represents a composite service. After the Web services composition is executed, the middleware sends back user a result.

5. **Optimization of Execution Time with Reinforcement Learning.** In this section, an HRL method for execution time optimization will be elaborated. Firstly, the

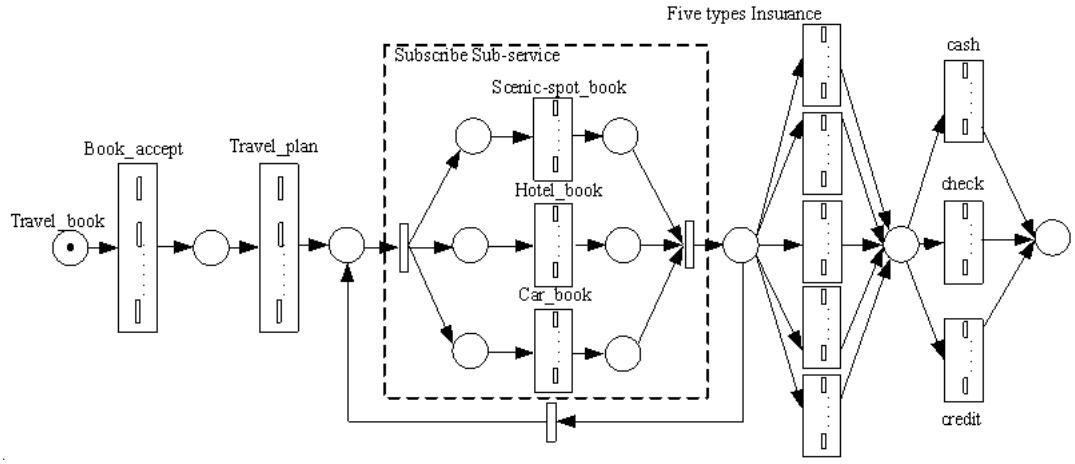


FIGURE 3. A Web services composition Petri net model

method of HRL is introduced. Secondly, the reward computation method of Q-learning (a kind of reinforcement learning) for elementary service and composite service is proposed. Lastly, an optimization algorithm for execution time is given.

5.1. **Reinforcement learning and hierarchical reinforcement learning.** Reinforcement Learning (RL) is an important machine learning method. RL finds the optimal action policy through the environment state observation and iterative trial of available actions. RL has the low quality for requirement in the prior knowledge and is also a kind of online learning method for the real-time environment. This characteristic adapts to Web services composition optimization [16]. An RL agent senses the environment and takes actions. It receives numeric rewards and punishments from some reward functions. Positive payoffs are calculated as rewards and negative payoffs are punishments. According to a policy function, the agent learns to choose adaptive actions to maximize a long term sum or average of the future payoffs it will receive [17].

When the same functional services increase and composition scales grow, the complexity of optimization will potentially increase as "the curse of dimensionality" of states when RL is applied. To reduce the complexity of optimization, we adopt a hierarchical reinforcement learning (HRL) to the Web services composition because services composition has a hierarchical architecture. The key principle underlying HRL is to develop learning algorithms that do not need to learn policies from scratch, but instead reuse existing policies for simpler subtasks [20,21]. Subtasks form the basis of hierarchical specifications of action sequences because they can include other subtasks in their definitions. The subtasks' value for RL is computed from its action sequences and the subtasks selection according to its value by RL.

5.2. **Learning and optimizing of execution time for elementary service.** In this paper, Q-learning, a kind of RL, is developed as HRL method. Because RL is based on Markov decision process (MDP), the relation of services composition and MDP is stated firstly.

1) The Markov decision process (MDP). The services composition process is an MDP. In services composition Petri net model, Petri net states set is looked as service states set $\boldsymbol{S}$, and transition set is looked as action set $\boldsymbol{WS}$. Here, a service's pre-state and post-state are $\boldsymbol{s}$ and $\boldsymbol{s}'$ respectively. $R_{ss'}^{ws}$ is the reward when changing state $\boldsymbol{s}$ to $\boldsymbol{s}'$ using action $\boldsymbol{ws}$ [18]. In Figure 4, when each place P_1, P_2, P_3 has a token respectively and other places do not have token, they are looked as state $\boldsymbol{s}$, $\boldsymbol{s}'$, $\boldsymbol{s}''$. When Web services

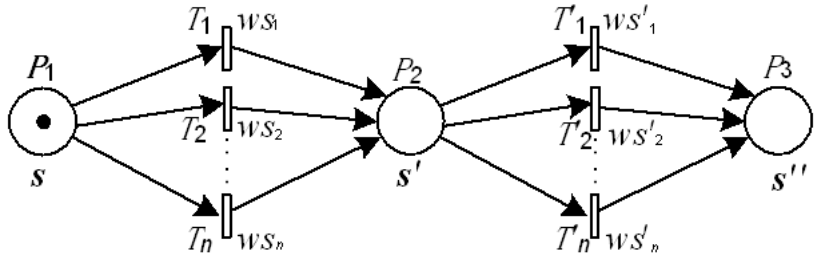


FIGURE 4. The selection model for same function services

composition is at state $\boldsymbol{s}$ and $\boldsymbol{s}$ is transferred to $\boldsymbol{s}'$ state by service action $\boldsymbol{ws}_1$, the reward value of $\boldsymbol{ws}_1$ is $R_{ss'}^{ws_1}$.

2) The Q value calculation in Q-learning. The service execution time is dynamic and it obeys exponential distribution. When a service action is selected and executed at i-times, the execution time t_i is accumulated and the service execution rate parameter λ of service execution time distribution can be evaluated. Now, if the total execution time of a service action is n, the execution time $t = (t_1, \ldots, t_n)$ is independent and the probability density is $\lambda \exp(-\lambda T)$. The likelihood function for λ is:

$$L(\lambda) = \prod_{i=1}^{n} \lambda \exp(-\lambda t_i) = \lambda^n \exp\left(-\lambda \sum_{i=1}^{n} t_i\right) = \lambda^n \exp(-\lambda n \overline{t}) \tag{10}$$

where $\overline{t} = \frac{1}{n} \sum_{i=1}^{n} t_i$. The derivative of the likelihood function's logarithm is:

$$\frac{d}{d\lambda} \ln L(\lambda) = \frac{d}{d\lambda} \left(n \ln(\lambda) - \lambda n \overline{t}\right) = \frac{n}{\lambda} - n\overline{t} = 0 \tag{11}$$

Consequently the maximum likelihood estimation for the rate parameter is:

$$\hat{\lambda} = \frac{1}{\overline{t}} \tag{12}$$

If a Web service action $\boldsymbol{ws}$ is selected n times and the execution changes its state $\boldsymbol{s}$ to the next state $\boldsymbol{s}'$, the reward of execution time is given by:

$$R_t^{ws} = \hat{\lambda} = \frac{1}{\overline{t}} \tag{13}$$

When the reward of service execution time is obtained, state-action value function $Q_t(\boldsymbol{s}, \boldsymbol{ws})$ is updated with Formula (14).

$$Q_t(\boldsymbol{s}, \boldsymbol{ws}) \leftarrow Q_t(\boldsymbol{s}, \boldsymbol{ws}) + \alpha \left[R_t^{ws} + \gamma \max_{ws'} Q_t(\boldsymbol{s}', \boldsymbol{ws'}) - Q_t(\boldsymbol{s}, \boldsymbol{ws})\right] \tag{14}$$

Here, α is a learning rate, γ is a damping rate.

3) The policy of services selection. After reward is gotten, the service selection policy will be studied using these values. The simplest service selection rule is to select the service with the highest value, i.e., the service corresponding to the maximum $Q_t(\boldsymbol{s}, \boldsymbol{ws})$. This service is called a greedy action. If a greedy action is selected, the learner (agent) exploits the current knowledge. If instead selecting one of the non-greedy actions, agent intends to explore to improve its policy. Exploitation is to do the right thing to maximize the expected reward on the one play; meanwhile, exploration may produce the greater total reward in the long run. Web service is executed in the dynamical environment. QoS of a service may be different among different executions. The QoS of a service needs to be grasped after long time learning [17]. A method using near-greedy selection rule called

ε-greedy method is used in Web services composition. Algorithm 1 shows how to realize elementary service selection using ε-greedy method of RL.

Algorithm 1. The Selection Algorithm for Elementary Web Service

Step 1. Initialization. When a service is executed at the first time, the $Q_t(\boldsymbol{s}, \boldsymbol{ws})$ is set as an arbitrarily constant for every service has equivalent chance being executed;

Step 2. Repeat from i) to iii) for each same function service selection until the composition is finished.

i) The selection policy for the composite service is ε-greedy (ε is set according to execution environment by user, usually $0 < \varepsilon \ll 1$);

A: Service with the biggest $Q_t(\boldsymbol{s}, \boldsymbol{ws})$ is selected at probability $1 - \varepsilon$;

B: Services is selected randomly at probability ε;

ii) Execute the selected service.

iii) Update the $Q_t(\boldsymbol{s}, \boldsymbol{ws})$ of the executed service according to (14).

5.3. **Optimization for execution time of composite services.** When a Web services composition is generated as mentioned in Section 4, there are several sub-services in a composite service. The composite service usually has a hierarchical architecture. The composite service in Figure 5 has two sub-services (service $\boldsymbol{A}$ and service $\boldsymbol{B}$). How to choose one from $\boldsymbol{A}$ or $\boldsymbol{B}$? In some pre-works, the exhaustive search is used. Here, a method which combines service function composition and QoS selections is proposed. The elementary service's QoS obtains through monitoring the service execution and the Q value of each elementary service is calculated. Q value for each composite sub-service, such as QoS of sub-service $\boldsymbol{A}$ in Figure 5, is calculated according to composite Flow Rule. These composite sub-services have four composition types. If these composite sub-services are composed by composite sub-service, the QoS and Q value of composite service is calculated hierarchically.

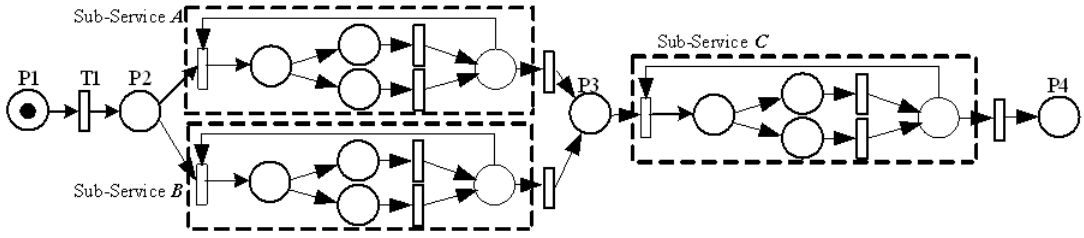


FIGURE 5. The model of two composite sub-services selection

In elementary service selection algorithm, the service execution rate λ is used as reward of RL. The reward of execution time for composite service is calculated as the following [12].

1) Sequential composition: If a composite service has N sequential sub-services and i-th sub-service has execution rate λ_i, the composite service is looked as a service action and reward is calculated by (15).

$$R_t^{ws} = \frac{1}{\sum_{i=1}^{N} \frac{1}{\lambda_i}} \tag{15}$$

2) Parallel composition: If a composite service has N parallel sub-services and i-th sub-service has execution rate λ_i, the composite service is looked as a service action and reward is calculated as (16).

$$R_t^{ws} = \min(\lambda_1, \lambda_2, \cdots, \lambda_N) \tag{16}$$

3) Alternative composition: If a composite service has N alternative sub-services and i-th sub-service has execution rate λ_i, the composite service is looked as a service action. If the service is free selected, the select method is the same selection policy of RL and the reward is calculated as below.

$$R_t^{ws} = \max(\lambda, \lambda_2, \cdots, \lambda_N) \tag{17}$$

4) Iterative composition: If a composite service is N times iteration of sub-services and service has execution rate λ, the composite service is looked as a service action. If we unfold the cyclic process, the flow is N times service sequential execution. The reward is calculated as below.

$$R_t^{ws} = \frac{1}{\sum_{i=1}^{N} \frac{1}{\lambda}} \tag{18}$$

After the R_t^{ws} of composite service is obtained, the action value can be updated using $Q_t(\boldsymbol{s}, \boldsymbol{ws}) \leftarrow Q_t(\boldsymbol{s}, \boldsymbol{ws}) + \alpha \left[R_t^{ws} + \gamma \max_{ws'} Q_t(\boldsymbol{s}', \boldsymbol{ws}') - Q_t(\boldsymbol{s}, \boldsymbol{ws}) \right]$ (Formula (14)). Here, $\boldsymbol{ws}'$ is a composite service action which is in same hierarchy with $\boldsymbol{ws}$. For example, in Figure 5, if sub-service $\boldsymbol{A}$ or $\boldsymbol{B}$ is looked as $\boldsymbol{ws}$, sub-service $\boldsymbol{C}$ is $\boldsymbol{ws}'$.

The algorithm of hierarchical services selection is given in Algorithm 2.

Algorithm 2. The Hierarchical Selection Algorithm for Web Service

Step 1. Initialization. $Q_t(\boldsymbol{s}, \boldsymbol{ws})$ is set as a arbitrary constant for each service has an equivalent chance being executed;

Step 2. When a user invocation comes, it is compared with the rules in the service flow rule set. If the invocation is a composite service, repeat from i) to iv);

i) If there are several same function composite services, one is selected using selection policy for the composite service. ε-greedy is used as the selection policy (ε is set according to execution environment by user, usually $0 < \varepsilon \ll 1$).

ii) The selected composite service is hierarchically decomposed into sub-service according to Web services composition rule.

iii) If the decomposed services are composite services, then use these composite services as an invocation and invoking Algorithm 2.

iv) Until the invocation decompose into elementary service invocations;

v) Calculate $Q_t(\boldsymbol{s}, \boldsymbol{ws})$ according to (14) and return $Q_t(\boldsymbol{s}, \boldsymbol{ws})$ as the composite service reward.

Step 3. If the invocation is an elementary service invocation, then invoking Algorithm 1.

The middleware gradually grasps the execution time of service after many times execution, and chooses the service according to the characteristic of execution time. As a result, the execution time of composite service is optimized.

5.4. **The complexity analysis.** In some pre-works, the exhaustive search is used for finding optimal service path [3]. Assuming that there are n service tasks and m candidate Web services for each task, the total number of execution paths is m^n. The computation cost is very high. Here, we propose to compose the services hierarchically. Suppose that each hierarchy has k sub-services. There are $\lceil \log_k m \rceil$ ($\lceil\ \rceil$ is upward to nearest integer) hierarchies. The lowest hierarchy has k^n paths and other $\lceil \log_k m \rceil^{-1}$ hierarchies have $k^{\lceil \log_k m \rceil^{-1}}$ paths. So, the total paths are $k^n * k^{\lceil \log_k^m \rceil^{-1}}$, i.e., $k^{n+\lceil \log_k^m \rceil^{-1}}$. Usually, each hierarchy only has several services tasks; the complexity of the QoS optimization model dramatically decreases according to the algorithm of HRL.

6. **Optimization for Multi-Dimensional QoS.** Execution time (a single-dimensional of QoS) optimization has been given in Section 5. In this section, the QoS optimization considering multi-dimensional QoS will be dealt with.

6.1. **Execution cost, usability optimization for elementary service.** When the services composition is executed, middleware attains the service's execution cost and usability of service. These services' QoS values are transformed as service action reward in Q-learning.

1) Execution cost reward. Because the lower value of execution cost is a better quality. The reward r_c^{ws} is calculated as $r_c^{ws} = 1/C_{service}$.

Because the cost is dynamical, we assume that the reward of i-th execution is $(r_c^{ws})_i$ and evaluation reward value after k time learning is R_c^{ws}. Then, the reward of a service's execution cost after k time execution is:

$$R_c^{ws} = \frac{1}{\frac{1}{k}\sum\limits_{i=1}^{k}\frac{1}{(r_c^{ws})_i}} \tag{19}$$

2) Usability reward. If a service action is i-th executed and $U_{service}$ is gotten using Formula (3), R_u^{ws} is given by Formula (20).

$$R_u^{ws} = U_{service} \tag{20}$$

After the R_c^{ws} and R_u^{ws} of elementary service are obtained, the action value is updated using (21) and (22).

$$Q_c(\boldsymbol{s}, \boldsymbol{ws}) \leftarrow Q_c(\boldsymbol{s}, \boldsymbol{ws}) + \alpha\left[R_c^{ws} + \gamma \max_{ws'} Q_c(\boldsymbol{s'}, \boldsymbol{ws'}) - Q_c(\boldsymbol{s}, \boldsymbol{ws})\right] \tag{21}$$

$$Q_u(\boldsymbol{s}, \boldsymbol{ws}) \leftarrow Q_u(\boldsymbol{s}, \boldsymbol{ws}) + \alpha\left[R_u^{ws} + \gamma \max_{ws'} Q_u(\boldsymbol{s'}, \boldsymbol{ws'}) - Q_u(\boldsymbol{s}, \boldsymbol{ws})\right] \tag{22}$$

The $Q_t(\boldsymbol{s}, \boldsymbol{ws})$ is changed with $Q_c(\boldsymbol{s}, \boldsymbol{ws})$ or $Q_u(\boldsymbol{s}, \boldsymbol{ws})$ in Algorithm 1 and the algorithm of optimization of execution time or usability will be gotten.

6.2. **Optimization policy for multi-dimensional QoS.** After the Q-value of service is learned, a method which can estimate the probability of every service trend to optimization of execution time, execution cost or usability can be founded. As shown in Figure 4, when $T_1, \ldots, T_n$ get state-action value of the execution time $Q_c(\boldsymbol{s}, \boldsymbol{ws}_1), \ldots, Q_c(\boldsymbol{s}, \boldsymbol{ws}_n)$, the service is selected by a Soft-Max method according to a probability of Gibbs distribution:

$$\Pr\{\boldsymbol{s}, \boldsymbol{ws}_i^t\} = \frac{e^{Q(s,ws_i)\beta}}{\sum_{ws\in A} e^{Q(s,ws)\beta}} \tag{23}$$

where $\Pr\{\boldsymbol{s}, \boldsymbol{ws}_i^t\}$ is a probability to select service $\boldsymbol{ws}_i$, β is a positive inverse temperature constant and $\boldsymbol{A}$ is a set of available services. In the same way, $\Pr\{\boldsymbol{ws}_i^c\}$, $\Pr\{\boldsymbol{ws}_i^u\}$ is determined.

In the case of only considering the execution time during the QoS optimization, the selected probability of i-th service is $\Pr\{\boldsymbol{ws}_i^t\}$. In the case of considering execution time or usability, the selection probability of i-th service comes to be given by $\Pr\{\boldsymbol{ws}_i^c\}$ or $\Pr\{\boldsymbol{ws}_i^u\}$. When the three-dimensional QoS is considered, an i-th service QoS optimization vector is defined as:

$$\Pr\{\boldsymbol{ws}_i\} = (\Pr\{\boldsymbol{ws}_i^t\}, \Pr\{\boldsymbol{ws}_i^c\}, \Pr\{\boldsymbol{ws}_i^u\}) \tag{24}$$

Because different users have different demands on these quality criteria, the different weights are set for each criterion and called credible degree. The vector of credible degree is

defined as $Credit = (\theta_t, \theta_c, \theta_u)$. The probability to select service $\boldsymbol{ws}_i$, which is considered for a multi-dimensional QoS, is:

$$\Pr(\boldsymbol{s}, \boldsymbol{ws}_i) = (\Pr\{\boldsymbol{ws}_i^t\}, \Pr\{\boldsymbol{ws}_i^c\}, \Pr\{\boldsymbol{ws}_i^u\}) \times (\theta_t, \theta_c, \theta_u)^T \tag{25}$$

When $\Pr(\boldsymbol{s}, \boldsymbol{ws}_i)$ is gotten, it is used in Algorithm 1 for service selection and the policy of optimization for multi-dimensional QoS is obtained.

6.3. **Hierarchical optimization for multi-dimensional QoS.** When a Web services composition is given by composite sub-services, the value of QoS for composite sub-services is the base of computing reward for HRL. Now, the rewards are calculated as below when a composite sub-service is executed. Here, the reward R_c^{ws} and R_u^{ws} of each elementary service is gotten as Formulae (19) and (20).

1) Sequential composition: If a composite service is composited by N sequential services, it is looked as a service action. The reward of execution cost and usability is:

$$R_c^{ws} = \frac{1}{\sum\limits_{i=1}^{N} \frac{1}{(R_c^{ws})_i}} \tag{26}$$

$$R_u^{ws} = \prod_{i=1}^{N} (R_u^{ws})_i \tag{27}$$

2) Parallel composition: If a composite service is composited by N parallel services, it is looked as a service action. The reward of execution cost and usability is:

$$R_c^{ws} = \frac{1}{\sum\limits_{i=1}^{N} \frac{1}{(R_c^{ws})_i}} \tag{28}$$

$$R_u^{ws} = \sum_{i=1}^{N} (R_u^{ws})_i - \sum_{i=1}^{N-1} \sum_{j=i-1}^{N} Q\,(R_u^{ws})_i\, Q\,(R_u^{ws})_j + \cdots + (-1)^N \prod_{i=1}^{N} (R_u^{ws})_i \tag{29}$$

3) Alternative composition: If a composite service is composited by N alternative services and the composite service is looked as a service action. The reward of execution cost and usability is:

$$R_c^{ws} = \max((R_c^{ws})_1, (R_c^{ws})_2, \ldots, (R_c^{ws})_N) \tag{30}$$

$$R_u^{ws} = \max((R_u^{ws})_1, (R_u^{ws})_2, \ldots, (R_u^{ws})_N) \tag{31}$$

4) Iterative composition: If a composite service is N times iteration of sub-services, the total iteration service actions service is looked as one service action. If we unfold the cyclic process, the flow is N times service sequential execution. The reward of execution cost and usability is:

$$R_c^{ws} = \frac{1}{\sum\limits_{i=1}^{N} \frac{1}{(R_c^{ws})_i}} \tag{32}$$

$$R_u^{ws} = \prod_{i=1}^{N} (R_u^{ws})_i \tag{33}$$

After the rewards R_t^{ws}, R_c^{ws} and R_u^{ws} are given, the hierarchical optimization for multi-dimensional QoS has three steps. Firstly, $Q_t(\boldsymbol{s}, \boldsymbol{ws})$, $Q_c(\boldsymbol{s}, \boldsymbol{ws})$, $Q_u(\boldsymbol{s}, \boldsymbol{ws})$ are calculated according to Formulae (14), (21), (22). Secondly, $\Pr\{\boldsymbol{ws}_i^t\}$, $\Pr\{\boldsymbol{ws}_i^c\}$, $\Pr\{\boldsymbol{ws}_i^u\}$ are computed using Formula (23). Lastly, $\Pr(\boldsymbol{s}, \boldsymbol{ws}_i)$ is gotten through (25). After getting the

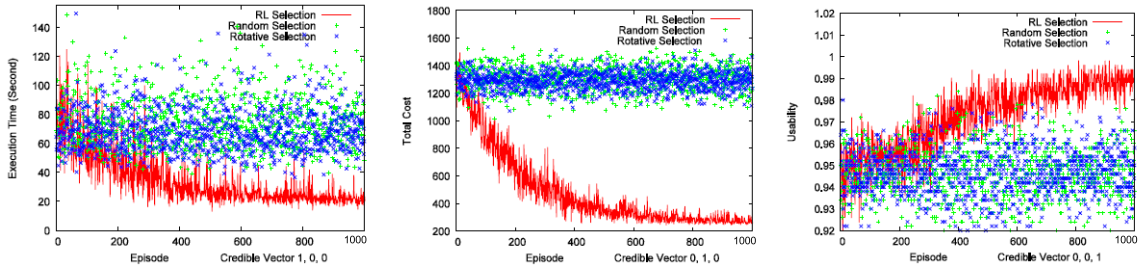
$\Pr(\boldsymbol{s}, \boldsymbol{ws}_i)$, it is used in Algorithm 2 for service selection and the policy of optimization for multi-dimensional QoS is obtained.

7. **The Experiment.** In order to evaluate the proposed approach, the QoS of a travel booking Web services composition, which is listed in Figure 3, is optimized as an example. In the experiment, Web services are created using Microsoft Visual studio .NET C# and deployed on local service. And services are registered in local UDDI offering by Windows 2003. The invocation, learning and composition program are created using Microsoft Visual studio .NET C++. The invocation messages are transmitted by SOAP ToolKit 3.0. The composition rules and QoS values are stored in SQL Server 2000 Database.

The Web services are composed according to the rules listed by (5)-(9) in Section 4.2. The subscribed sub-service is executed 10 times for booking 10 different scenic-spots. 16 same function services are created for each elementary service in Figure 3. Each of 16 same function services has a execution rate λ_i $(i = 1, 2, \ldots, 16)$ and has a time-lapse T which obeys to Formula (1). Before starting execution of the services composition, the composition rules according to business processes are stored into Database. When middleware accepts a user's invocation, the services are composed according to composition rules. Rewards of elementary service and composite sub-service are recorded into Database after learning from the services execution.

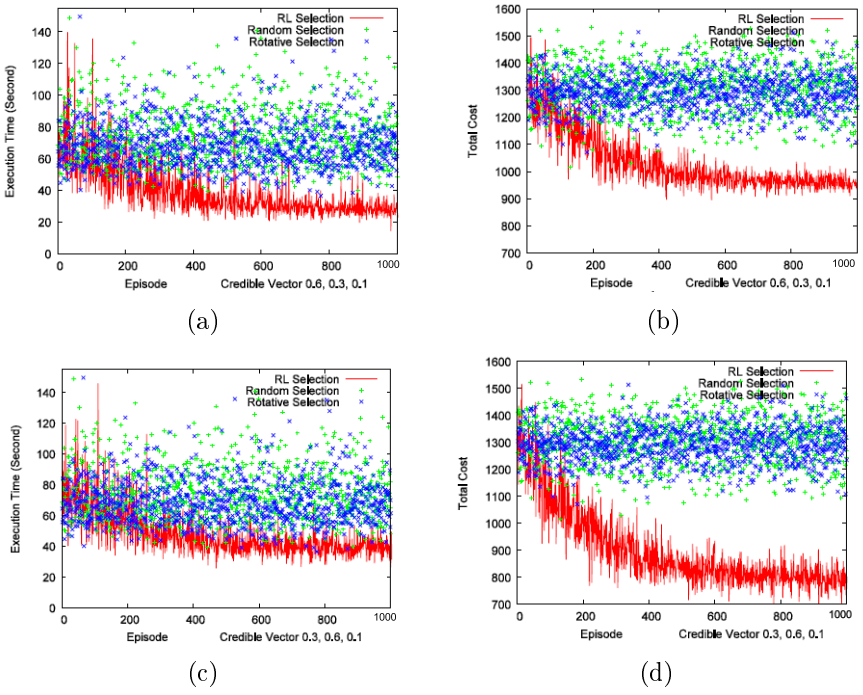
7.1. **Web services composition optimization for single-dimensional QoS.** The services are selected according to one of execution time, execution cost and usability. When optimization of the execution time is only considered, the elementary and composite services are selected according to $Q_t(\boldsymbol{s}, \boldsymbol{ws})$. When optimization of the execution cost is only considered, the elementary and composite service are selected according to $Q_c(\boldsymbol{s}, \boldsymbol{ws})$. When optimization of the usability is only considered, the elementary and composite service are selected according to $Q_u(\boldsymbol{s}, \boldsymbol{ws})$. Here, α and γ are set as 0.1 and 0 in Formulae (14), (21), (22). The travel booking composite service is executed 1,000 times and the elementary and composite services are selected by 3 methods: random, rotation and HRL. The execution time, execution cost, usability data are shown in Figure 6. The data indicate that HRL algorithm modified the value of execution time, execution cost and usability respectively. The optimal path of services execution time, execution cost and usability were selected. The data listed in Figure 6 show that the middleware had obtained the optimized QoS of services and found the optimal execution path after 400 times execution. The execution time, execution cost and usability of travel booking services composition were dramatically optimized.

7.2. **Web services composition optimization for multi-dimensional of QoS.** The travel booking composite service is executed 1,000 times and the elementary and composite services are selected by 3 methods: random, rotation and HRL. All elementary services have 16 candidate services. We show the execution time and execution cost in Figure 7 where vector of credible degree $Credit = \{0.6, 0.3, 0.1\}$ and $Credit = \{0.3, 0.6, 0.1\}$ were used in Formula (25) corresponding to execution time, execution cost and usability. The data indicate that the middleware optimized multi-dimensional QoS and modified $Q(\boldsymbol{s}, \boldsymbol{ws})$ according to credible degree. Comparing Figures 7(a) and 7(c), the optimization of execution time in (a) is better than in (c) because the execution time credible degree in (a) is higher than in (c). Comparing Figures 7(b) and 7(d), the optimization of execution cost in (b) is better than in (d) because the execution cost credible degree in (d) is higher than in (b). So, it is showed that the middleware not only can optimize the QoS but also can differently optimize execution time, execution cost and usability according to credible degree.



(a) (b) (c)

FIGURE 6. (a) The execution time, (b) execution cost and (c) usability comparison by different services selection methods. Here, only one dimension QoS is considered, so the credible vector $(\theta_t, \theta_c, \theta_u)$ were set as $(1, 0, 0)$, $(0, 1, 0)$ or $(0, 0, 1)$.



(a) (b) (c) (d)

FIGURE 7. The executions time and execution cost comparison using different service selection methods. (a) and (c) are the execution time comparison when credible vector $(\theta_t, \theta_c, \theta_u)$ is $(0.6, 0.3, 0.1)$ and $(0.3, 0.6, 0.1)$ respectively. (b) and (d) are the service cost comparison when credible vector $(\theta_t, \theta_c, \theta_u)$ is $(0.6, 0.3, 0.1)$ and $(0.3, 0.6, 0.1)$ respectively.

7.3. **Comparison between web services composition optimization methods.** Zeng et al. has proposed the QoS optimization approach using Integer Programming (IP) [3]. And Florian et al. have proposed the QoS optimization approach using the heuristic method of Simulated Annealing (SA) [4]. We use these methods in our experiment environment. Here, there are 50 service tasks being composed sequentially and every task has 16 candidate services. There are three service selection methods which are used in each simulation. The three service selection methods are HRL which is proposed in this paper, conventional which is IP or SA, and random selection. Here, we set the credible vector as $\{1, 0, 0\}$ and $\{0, 1, 0\}$ for each method, i.e., only consider execution time and execution

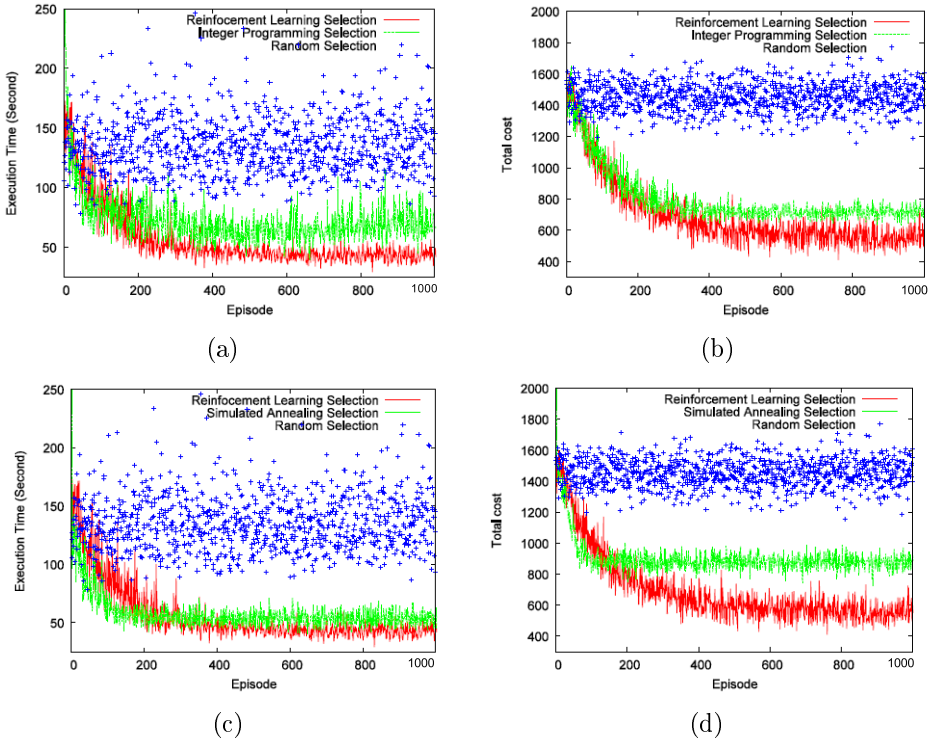


FIGURE 8. The executions time and execution cost comparison for different service selection methods. (a) and (b) are the execution time and service cost comparison with RL, IP and random when credible vector $(\theta_t, \theta_c, \theta_u)$ is $(1, 0, 0)$ and $(0, 1, 0)$ respectively. (c) and (d) are the execution time and service cost comparison with RL, SA and random when credible vector $(\theta_t, \theta_c, \theta_u)$ is $(1, 0, 0)$ and $(0, 1, 0)$ respectively.

cost in simulations. We get execution results for execution time data in Figure 8(a) and 8(c), execution cost data in Figure 8(b) and 8(d). The result of simulation shows that HRL method is better than conventional IP and SA methods after 300 times execution.

8. **Conclusions and Future Work.** We proposed a middleware which realizes Web services composition and QoS optimization of Web service using a Petri net and HRL algorithms. An extensible QoS model, the value of QoS of which is obtained by the learning from the service execution, is founded. In this QoS model, the QoS of elementary service is grasped by the middleware when the service is executed. As Web services composition is a hierarchically architecture, we constructed the hierarchically stochastic Petri net model of Web services composition. After the middleware attains QoS of elementary services, the QoS of composite service is optimized according to HRL algorithms. Single-dimensional and multi-dimensional optimization for elementary service and composite sub-service are considered in the middleware. We used the proposed method to study a real Web service problem (the travel booking system) and the simulation results showed the effectiveness of QoS optimization with HRL. We plan to extend our work to the real deployment of the method on the Web and to rich the QoS models by service requesters.

REFERENCES

[1] Y.-P. Chen and Z.-Z. Li, E-WsFrame: A framework support QoS driven web services composition, *Information Technology Journal*, vol.6, no.3, pp.390-395, 2007.

[2] M. Ramakanta et al., Web-services classification using intelligent techniques, *Expert Systems with Applications*, vol.37, no.7, pp.5484-5490, 2010.
[3] L. Zeng, B. Benatallah et al., QoS-aware middleware for web services composition, *IEEE Transactions on Software Engineering*, vol.30, no.5, pp.311-327, 2004.
[4] R. Florian et al., Metaheuristic optimization of large-scale QoS-aware service compositions, *Proc. of the IEEE International Conference on Services Computing*, pp.97-104, 2010.
[5] H. Wang and P. Tang, Preference-aware web services composition by reinforcement learning, *IEEE International Conference on Tools with Artificial Intelligence*, pp.379-386, 2008.
[6] Y. Zhang, H. Huang, D. Yang and H. Zhang, A hierarchical and chord-based semantic service discovery system in the universal network, *International Journal of Innovative Computing, Information and Control*, vol.5, no.11(A), pp.3745-3753, 2009.
[7] Y. Liu, A. H. H. Ngu and L. Zeng, QoS computation and policing in dynamic web service selection, *Proc. of the 13th International World Wide Web Conference on Alternate Track Papers & Posters Table of Contents*, New York, NY, USA, pp.66-73, 2004.
[8] B. Orriens, J. Yang and M. P. Papazoglou, A framework for business rule driven web services composition, *Lecture Notes in Computer Science*, pp.52-64, 2004.
[9] D. Daniela et al., Automatic composition of transition-based semantic web services with messaging, *Proc. of the 31st International Conference on Very Large Data Bases*, Trondheim, Norway, pp.613-624, 2005.
[10] S. Hwang, E. Lim, C. Lee and C. Chen, Dynamic web service selection for reliable web services composition, *IEEE Transactions on Services Computing*, vol.1, no.2, pp.104-116, 2008.
[11] Y. Achbany, I. J. Jureta, S. Faulkner and F. Fouss, Continually learning optimal allocations of services to tasks, *IEEE Transactions on Services Computing*, vol.1, no.3, pp.141-154, 2008.
[12] A. Martens, Usability of web services, *Proc. of the 4th International Conference on Web Information System Engineering Workshop IEEE*, pp.182-190, 2004.
[13] D. A. Menasce, Response-time analysis of composite Web service, *IEEE Internet Computing*, vol.8, no.1, pp.90-92, 2004.
[14] P. Wohed, W. M. P. Aalst et al., Analysis of web services composition languages: The case of BPEL4WS, *The 22nd International Conference on Conceptual Modeling*, Springer-Verlag, Berlin Heidelberg, pp.200-215, 2003.
[15] H. Rachid and B. Boualem, A petri net-based model for web services composition, *Proc. of 14th Australasian Database Conference*, Adelaide, Australia, pp.191-200, 2003.
[16] J. Yao, C. K. Tham and K. Y. Ng, Decentralized dynamic workflow scheduling for grid computing using reinforcement learning, *The 14th IEEE International Conference on Networks*, pp.1-6, 2006.
[17] R. S. Sutton and A. G. Batto, *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, MA, USA, 1998.
[18] P. Doshi, R. Goodwin and R. Akkiraju, Dynamic workflow composition using Markov decision processes, *Proc. of the IEEE International Conference on Web Service*, pp.576-582, 2004.
[19] R. Gronmo and C. J. Michael, Model-driven semantic web services composition, *Proc. of the 12th Asia-Pacific Software Engineering Conference IEEE*, pp.1-8, 2005.
[20] M. Ghavamzadeh and S. Mahadevan, Hierarchical average reward reinforcement learning, *Journal of Machine Learning Research*, no.8, pp.2629-2669, 2007.
[21] Y. Hirashima, A Q-learning system for container transfer scheduling based on shipping order at container terminals, *International Journal of Innovative Computing, Information and Control*, vol.4, no.3, pp.547-558, 2008.