# A Novel Approach to Large-Scale Services Composition

Hongbing Wang* and Xiaojun Wang

School of Computer Science and Engineering, Southeast University, Nanjing, China
{hbw,seuwxj}@seu.edu.cn

**Abstract.** We investigate a multi-agent reinforcement learning model for the optimization of Web service composition in this paper. Based on the model, a multi-agent Q-learning algorithm was proposed, where agents in a team would benefit from one another. In contrast to single-agent reinforcement-learning, our algorithm can speed up the convergence to optimal policy. In addition, it allows composite service to dynamically adjust itself to fit a varying environment, where the properties of the component services continue changing. A set of experiments is given to prove the efficiency of the analysis. The advantages and the limitations of the proposed approach are also discussed.

**Keywords:** Web Service composition, multi-agent.

## 1 Introduction

As a common understanding, Web services are self-describing and open building blocks for rapid, low-cost composition of distributed applications [8]. In practice, a single Web service may not be sufficient at performing complex tasks. We usually need to combine multiple existing services together to meet customers' complex requests. With today's SOC technology, such composition is usually performed by human engineers. However, the Web environment is highly dynamic, and most Web services are evolving at all time. A service engineer cannot always foresee all the changes that could happen in the future. A manual service composition can also be too rigid to adapt to a dynamic environment. Therefore, dynamic service composition is regarded as a crucial functionality for the Web of the future. Different technologies of computational intelligence have been investigated for solving the problem of dynamic service composition.

AI planning is a typical type of techniques used to automate Web services composition [9] [2]. Doshi [3] and Gao [4] have studied the application of MDPs (Markov Decision Processes) in Web service composition. MDPs assume a fully observable world and require explicit reward functions and state transition functions. Such requirements are too strict to a real world scenario. Wang et al. [11] proposed to use reinforcement learning (RL) for service composition, so as to

* This work is partially supported by NSFC (61232007) and JSNSF of China (No.BK2010417).

Y. Ishikawa et al. (Eds.): APWeb 2013, LNCS 7808, pp. 220–227, 2013.

avoid complex modeling of the real world. Although it has been proved to be effective for small scale service compositions, it can be overly computationally expensive when working on a large number of services.

In this paper, we present a novel mechanism based on multi-agent reinforcement learning to enable adaptive service composition. The model proposed in this paper extends the reinforcement learning model that we have previously introduced in [11]. In order to reduce the time of convergence, we introduce a sharing strategy to share the policies among the agents, through which one agent can use the policies explored by the others. As the learning process continues throughout the life-cycle of a service composition, the composition can automatically adapt to the change of the environment and the evolvement of its component services. Experimental evaluation on large scale service compositions has demonstrated that the proposed model can provide good results.

## 2 A MAMDP Model for Service Composition

In this section we present the reinforcement learning model that we have previously introduced in [11] for solving the dynamic web service composition problem. The RL model introduced in [11] will be extended in this section towards a distributed architecture.

Reinforcement learning is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment [7]. One key aspect of reinforcement learning is a trade-off between exploitation and exploration. To accumulate a lot of reward, the learning system must prefer the best experienced actions, however, it has to try new actions in order to discover better action selections for the future. One such method is $\epsilon-$ greedy, when the agent chooses the action that it believes has the best long-term effect with probability $1-\epsilon$, and it chooses an action uniformly at random, otherwise.

We formally define the key concepts used in the model.

**Definition 1.** ***(Web Service).*** *A Web service is modeled as a triple* $WS =< Pr; E; QoS >$*,where*

- *Pr represents the precondition of* $WS$*, which specifies the states of the world in which* $WS$ *can be executed.*
- *E represents the effect of* $WS$*, which describes how* $WS$ *changes the state of the world.*
- *QoS is a n-tuple* $< att_1; att_2; :::; att_n >$*, where each* $att_i$ *denotes a QoS attribute of* $WS$*.*

As mentioned earlier, we use Multi-agent Markov Decision Process (MAMDP) to model service composition. A MAMDP involves multiple actions and paths for each agent to choose. For agent $m$, we call our service composition model WSC-MDP, which simply replaces the actions in a MDP with Web services.

**Definition 2.** ***(Web service composition MDP (WSC-MDP).*** *A Web service composition MDP is a 7-tuple WSC-MDP* $=< m, s_0^m, S^m, A^m(.), P^m, R^m, s_r^m >$*, where*

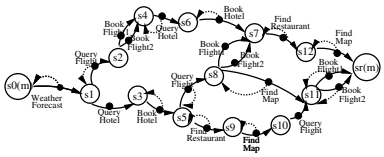


**Fig. 1.** The WSC-MDP of a Composite Service for Travel Plan

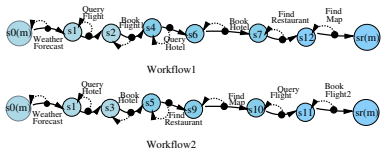


**Fig. 2.** Two Workflows Contained by the WSC-MDP in Fig. 1

- $m$*: denote the agent* $m$*.*
- $S^m$*: a finite set of state spaces of agent* $m$*.*
- $s_0^m \in S_m$*: is the initial state of agent* $m$ *and also an execution of the service compositions starts from this state.*
- $s_r^m \in S_m$*: is the set of terminal states for agent* $m$*. Upon arriving at one of the states, an execution of the service composition terminates.*
- $A^m(.)$*: represents the set of Web services that can be executed in state* $s \in S^m$ *. A service* $ws$ *belongs to* $A^m$*, only if the precondition* $ws$ $P$ *is satisfied by* $s$*.*
- $P^m : [p_{iaj}^m]$ $S^m \times A^m \times S^m \to [0, 1]$*: The transitions function for the agent. It expresses the probability that the agent* $m$ *goes to state* $j$ *if it executes web service* $a$ *in state* $i$ *is* $p_{iaj}^m$*.*
- $R^m : [r_{iaj}^m]$ $S^m \to R$*: defines the rewards that agent* $m$ *receives if it is in state* $i$ *and goes to state* $j$ *with the execution of web service* $a$*.*

A WSC- MDP can be visualized as a transition graph. As illustrated by Fig. 1, the graph contains two kinds of nodes, i.e. state nodes and service nodes, which are represented by open circles and solid circles respectively. $s0(m)$ represents the initial state node. The terminal states nodes are $sr(m)$. A state node can be followed by a number of service nodes, representing the possible services that can be invoked in the state. There is at least one arrow pointing from a service node to the next state node. Each arrow is labeled with a transition probability $p_{iaj}^m$, and the expected reward for that transition $r_{iaj}^m$. (For simplicity, we omit the labels in Fig. 1.) The transition probabilities on the arrows rooted at a single action node always sum to one.

**Definition 3.** *Service Workflow. Let* $wf$ *be a subgraph of a WSC-MDP.* $wf$ *is a service workflow if and only if there is at most one service that can be invoked at each state* $wf$*. In other words, a service workflow is actually equivalent to a deterministic state machine. A tradition service composition usually builds on a single such workflow.*

*Example 1.* Fig. 2 shows two of multiple service workflows. Which workflow to be executed is determined by the policy of the Markov decision process.

**Definition 4.** *(Policy): A policy* $\pi$ *is a mapping from state* $s \in S$ *to a service* $ws \in A$*, which tells which service* $ws = (s)$ *to execute when in state* $s$*.*

Each policy of a WSC-MDP can determine a single workflow. By executing a workflow, the service customer is supposed to receive a certain amount of reward, which is equivalent to the cumulative reward of all the executed services. Given a WSC-MDP, the task of our service composition system is to identify the optimal policy or workflow that offers the best cumulative reward. As the environment of a service composition keeps changing, the transition function P and the reward function R of a WSC-MDP change too. As a result, the optimal policy changes with time. If our system is able to identify the optimal policy at any given time, the service composition will be highly adaptive to the environment.

## 3 Algorithm for Service Composition

The proposed Distributed Approach

The Q-learning is the most popular and seems to be the effective model-free algorithm about the RL problems. It does not, however, address any of the issues involved in generalizing over large state and/ or action spaces [7]. That is why, in order to speed up the training process, we extend the proposed approach towards a distributed one, in which multiple cooperative agents learn to coordinate in order to find the optimal policy in their environment.

Experience sharing can help agents with similar tasks to learn faster and better. For instance, agents can exchange information using communication [10]. Furthermore, by design, most multiagent systems also allow the easy insertion of new agents into the system, leading to a high degree of scalability [1].

From agent-m's standpoint, its control task could be thought of as an ordinary reinforcement problem except that their action selecting strategy is dependent on other agent's optimal policies at the beginning of the learning. So at a certain state, one agent's policy may be useful to other agents which can help them to find optimal strategy quickly. But assume that each agent can simultaneously send its current policy to other agents, the communication information is huge. For the purpose of reducing the communication information we don't let the agent to communicate with each others, but introduce supervisor agent which supervises the learning process and synchronizes the computations of the individual agents. In our algorithm, each agent $m$ use the global Q-values estimations stored in the blackboard which stores the global Q-values estimations and communicate to the supervisor agent their intention to update a Q-value estimation.

So we have two types of agents in our architecture:

- WSCA (Web Service Composition Agents). Each WSCA agent runs in a separate process or thread and is trained using the Q-learning algorithm. Each local agent performs local Q-values estimations updating from its own point of view.
- a WSCS(Web Service Composition Supervisor) agent which supervises the learning process and synchronizes the computations of the individual WSCA

agents. It keeps a blackboard [5] which stores the global Q-values estimations. The local WSCA agents use the global Q-values estimations stored in the blackboard and communicate to the WSCS agent their intention to update Q-value estimation. If a local agent tries to update a certain Q-value, the WSCS agent will update the global Q-value estimation only if the new estimation received from the local agent is greater than the Q-values estimations existing in the blackboard.

In this study, each reinforcement-learning agent uses the one-step Q-learning algorithm. Its learning decision policy is determined by the central Q-table and the state/action value function which estimates long-term discounted rewards for each state/action pair. In order to get more knowledge for the agent and jump out of the sub-superior strategy trap, searching strategy was introduced into the Q-learning. The agent is allowed to take the action which isn't the superior at the current view, so $\varepsilon - greedy$ strategy was proposed. Thus the agent can explore the state-action space by choose the viable action randomly at some degree, and avoid arriving at the local superior solution via only choosing the action with the maximal Q-value.

For agent $m$, given a current state $s$ and available actions $A(m)$, a Q-learning agent selects action $a_i$ with a probability given by the rule below:

$$p^m(a_i|s) = \begin{cases} (1-\varepsilon) \text{ if } a_i = argMax_a\ Q[s,a] \\ \varepsilon \qquad\qquad others \end{cases}$$

In each time step, the agent $m$ updates $Q(s,a)$ by recursively discounting future utilities and weighting them by a positive learning rate $\alpha$ :

$$Q(s,a) = (1-\alpha) \cdot Q(s,a) + \alpha \cdot [r(s,a) + \gamma \cdot maxQ(s\prime, a\prime)]. \tag{1}$$

Here $\gamma(0 \leq \gamma < 1)$ is a discount parameter.

If $Q(s,a)$ is the biggest among the actions in state $s$, than the WSCS will update $Q(s,a)$ in the blackboard.

The training process consists of three phases and will be briefly described in the following.

Phase 1. Initial phase

The WSCS supervisor agent initializes the Q-values from the blackboard.

Phase 2. Training phase of each WSCA agent

During some training episodes, the individual WSCA agents will experiment some paths from the initial to a final state, using the $\epsilon - greedy$ mechanism and updating the Q-values estimations according to the algorithm described below (algorithm 1). We denote in following by $Q(s,a)$ the Q-value estimate associated to the state $s$ and $a$, as stored by the blackboard of the WSCS agent.

**Algorithm 1.** Multiagent Q-learning Algorithm for Agent $m$

**Require:**
The WSC-MDP for agent $m$;
The WSCS agent;
**repeat**
evaluate the starting state $s$
select action $a$ from $s$ using policy derived from $Q$ $(\epsilon - Greddy)$
**repeat**
**Learning:**(for each step of episode)
Take action $a$, observe the reward $r(s,a)$ and the next state $s\prime$.
WSCA agent asks WSCS agent for $Q(s,a)$.
WSCS retrieves $Q(s,a)$ from the blackboard.
WSCS sends the retrieved $Q(s,a)$ to WSCA.
WSCA agent updates the table entry $Q(s,a)$ ad follows

$$Q(s,a) = (1-\alpha) \cdot Q(s,a) + \alpha \cdot [r(s,a) + \gamma \cdot maxQ(s\prime, a\prime)]. \quad (2)$$

WSCA sends the new $Q(s,a)$ to WSCS.
WSCS updates the $Q(s,a)$ if it is greater than the old.
WSCA agent go to state $s\prime$
**until** $s$ is the terminal state
**until** the Q-values tiny changes

Phase 3. Executing phase

After the training of the multi-agent system has been completed, the solution learned by the WSCS supervisor agent is constructed by starting from the initial state and following the *Greedy* mechanism until a solution is reached. The system applies the solution as a service workflow to execute. At the same time, the execution is also treated as an episode of the learning process. The Q-functions are updated afterwards, based on the newly observed reward.

By combining execution and learning, our framework achieves self-adaptively automatically. As the environment changes, service composition will change its policy accordingly, based on its new observation of reward. It does not require prior knowledge about the QoS attributes of the component services, but is able to achieve the optimal execution policy through learning.

## 4 Experimental Evaluation

In order to evaluate the methods, we conducted simulation to evaluate the properties of our service composition mechanism based on the methods discussed in this paper. The PC configuration: Intel Xeon E7320 2.13GHZ with 8GB RAM, Windows 2003, jdk1.6.0.

We considered two QoS attributes of services. They ware service fee and execution time. We assigned each service node in a simulated WSC-MDP graph with random QoS values. The value followed normal distribution. To simulate

the dynamic environment, we periodically varied the QoS values of existing services based on a certain frequency. We applied the algorithms introduced in Section 3 to execute the simulated service compositions. The reward function used by the each learner was solely based on the two QoS attributes. After an execution of a service , the learners get a reward , whose value is:

$$R(s) = \frac{fee_i^{max} - fee_i^s}{fee_i^{max} - fee_i^{min}} + \frac{time_i^{max} - time_i^s}{time_i^{max} - time_i^{min}}$$

The reward was always positive, however service consumers always prefer low execution time and service fee.

We will show that such cooperative agents can speed up learning, measured by the average cumulative values in training, even though they will eventually reach the same asymptotic performance as independent agents.

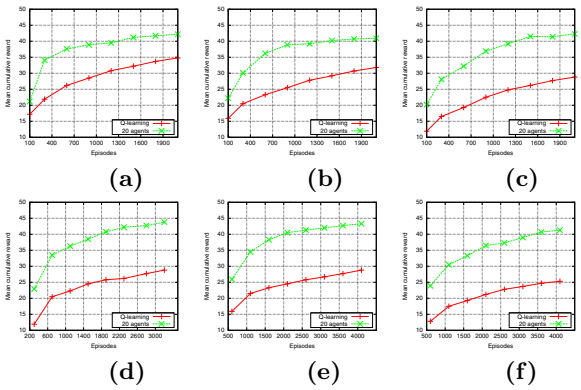


**Fig. 3.** (a) Results of comparison with 20 services in each state; (b) Results of comparison with 30 services in each state; (c) Results of comparison with 40 services in each state; (d) Results of comparison with 100 services in each state; (e) Results of comparison with 150 services in each state; (f) Results of comparison with 200 services in each state;

Scalability with respect to the number of services

In this stage of our evaluation, we studied the effect of distributed RL approach with varied number of services in each state. We fix the state number on 500 and vary the services from 20 to 200. As the Fig. 3 shows, the distributed RL learns more quickly than Q-learning, and when the number of services increases, the reducing of convergence time is more considerable, because they may have explored the different parts of a state space and share their knowledge. If agents perform the similar task, two agents can complement each other by exchanging their policies or use what the other agent had already learned for its own benefit. Assume that each agent can simultaneously send its current policy at some state to blackboard Q-table by WSCS agent, if some agent finds a better choice, it may update blackboard Q-table through WSCS agent, then other agents can adopt that policy with certain probability in that state.

The results in all cases clearly indicate that distributed RL approach presented in this paper learns more quickly and reduces the overall computational time compared againest the Q-learning.

## 5 Conclusion

This paper studied a novel framework for large scale service composition. In order to reduce the time of convergence we introduce a sharing strategy to share the policies among agents in a team. The experimental results show that the strategy of sharing state-action space improves the learning efficiency significantly. Additionally, the problem that has to be further investigated is how to reduce the communication cost between the WSCA agents and WSCS agent and explore other local search mechanisms. Next, we will concentrate on these issues and improve our algorithm further.

## References

1. Busoniu, L., Babuska, R., De Schutter, B.: A comprehensive survey of multiagent reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 38(2), 156–172 (2008)
2. Carman, M., Serafini, L., Traverso, P.: Web service composition as planning. In: ICAPS 2003 Workshop on Planning for Web Services, pp. 1636–1642 (2003)
3. Doshi, P., Goodwin, R., Akkiraju, R., Verma, K.: Dynamic workflow composition using markov decision processes. In: IEEE International Conference on Web Services, pp. 576–582. IEEE (2004)
4. Gao, A., Yang, D., Tang, S., Zhang, M.: Web service composition using markov decision processes. In: Fan, W., Wu, Z., Yang, J. (eds.) WAIM 2005. LNCS, vol. 3739, pp. 308–319. Springer, Heidelberg (2005)
5. Gonzaga, T., Bentes, C., Farias, R., de Castro, M., Garcia, A.: Using distributed-shared memory mechanisms for agents communication in a distributed system. In: Seventh International Conference on Intelligent Systems Design and Applications, ISDA 2007, pp. 39–46. IEEE (2007)
6. Hwang, S.Y., Lim, E.P., Lee, C.H., Chen, C.H.: Dynamic web service selection for reliable web service composition. IEEE Transactions on Services Computing 1(2), 104–116 (2008)
7. Kaelbling, L., Littman, M., Moore, A.: Reinforcement learning: A survey. Arxiv preprint cs/9605103 (1996)
8. Papazoglou, M., Georgakopoulos, D.: Service-oriented computing. Communications of the ACM 46(10), 25–28 (2003)
9. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: Htn planning for web service composition using shop2. Web Semantics: Science, Services and Agents on the World Wide Web 1(4), 377–396 (2004)
10. Sutton, R., Barto, A.: Reinforcement learning. Journal of Cognitive Neuroscience 11(1), 126–134 (1999)
11. Wang, H., Zhou, X., Zhou, X., Liu, W., Li, W., Bouguettaya, A.: Adaptive service composition based on reinforcement learning. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 92–107. Springer, Heidelberg (2010)