Service Selection for Service-Oriented Architecture using Off-line Reinforcement Learning in Dynamic Environments

Yuya Kondo and Ahmed Moustafa
Nagoya Institute of Technology, Japan

Keywords: Reinforcement Learning, Offline Reinforcement Learning, Transfer Learning.

Abstract: Service-Oritented Architeture (SOA) is a style of system design in which the entire system is built from a combination of services, which are functional units of software. The performance of a system designed with SOA depends on the combination of services. In this research, we aim to use reinforcement learning for service selection in SOA. Service selection in SOA is characterized by its dynamic environment and inefficient collection of samples for training. We propose an offline reinforcement learning method in a dynamic environment to solve this problem. In the proposed method, transfer learning is performed by applying fine tuning and focused sampling. Experiments show that the proposed method can adapt to dynamic environments more efficiently than redoing online reinforcement learning every time the environment changes.

1 INTRODUCTION

Service oriented computing (SOC) (Papazoglou, 2003) is computing paradigm that utilize service as fundamental elements for developing applications/solutions. The aim of SOC is to enable the interoperability among different software and data applications running of a variety of platforms. Service oriented architecture (SOA) is is one of method to create large scale computer system which consists of multiple services across network or platform. The benefit of SOA is high flexibility in system development. After each service is created, the system is assembled in SOA. If there is an inconvenient service during the assembly process, it is possible to modify that service and replace it with another service. As the number of options expands with the number of combinations, the system can be developed more flexibly. We can pick and choose what we need from the services we have created in the past, combine them, and add or recreate only the parts we need. This is another major advantage of SOA.

In SOA, multiple services are used to fulfill given task. Decision making in selecting service component to fulfill given task is needed. Reinforcement learning is good approach to solve this problems, and there are several related works. (Wang et al., 2020) (Moustafa and Ito, 2018) The aim of reinforcement learning is optimizing agent's action sequence during the interaction with learning environment. Reward is given when agent do good behavior to fulfill the goal. Agent learns good behavior by maximizing total reward. Game AI, recommend system and automated driving are the examples of application of reinforcement learning. (Silver et al., 2017) (Ie et al., 2019) (Wang et al., 2018) There are two major problems in using reinforcement learning for service selection in SOA.

- The cost of learning interactively with real-world environments is high.
- Service selection in SOA is a dynamic environment.

It requires huge number of trials for reinforcement learning. The cost of learning by constructing system in the real environment and getting feedback is high. Most conventional reinforcement learning methods assume a static environment. Therefore, it is difficult to apply them to real-world problems that deal with dynamic environments.

In this study, we propose an offline RL (Levine et al., 2020) method that is robust to dynamic environments. We improve the efficiency of relearning in exchange for some accuracy by reusing previous learning results for relearning when the environment changes and intentionally biasing the sampling of the dataset used for relearning. The proposed method addresses the problem of conventional methods which require relearning from the beginning every time the environment changes in a dynamic environment. We

Kondo, Y. and Moustafa, A.
Service Selection for Service-Oriented Architecture using Off-line Reinforcement Learning in Dynamic Environments.
DOI: 10.5220/0010872400003116
In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART 2022) - Volume 1*, pages 64-70
ISBN: 978-989-758-547-0; ISSN: 2184-433X

validate the proposed method through simulations of the SOA system construction. For real-world applications, the proposed method is effective in problem settings such as SOA system construction, where a dynamic environment and AI with strict accuracy are not required.

2 PRELIMINARIES

2.1 Service Oriented Architecture

These days, Computer System spreads to everywhere around us. It is impossible to control all of computer in one place. So distributed control is needed. These computers consist of various type of platform and device and each computer has their own function. Linking these function across devices and platforms has a potential to create good and large scale service. Service Oriented Architecture (SOA) is one of method to create large scale computer system which consist of multiple services across network or platform.

2.2 Deep Reinforcement Learning

Reinforcement learning (RL) (Sutton and Barto, 2018) is optimization for action sequence in given environment. Agent learns behavior to fulfill the goal during the interaction with environment. Agent decides its behavior based on policy. Agent can recognize various information from environment, this information treat as state. After agent take some action, agent moves to next state due to change of information given from environment. This single process is called step and this is minimum unit of agent's behavior. Agent repeats this process until agent moves from initial state to terminal state. This sequential process is called episode. In shooting game, one step is one minimum recognizable frame and one episode is a flow from the start to end of game. To evaluate behavior, scalar value is used and this value is called reward. Agent acts in environment and get reward. Agent improves the policy based on reward. In reinforcement learning, markov decision process (MDP) is used to define learning environment. MDP consists of state, reward, action and transition probability.

Q-Learning: Q-Learning (Watkins and Dayan, 1992) is classic method of reinforcement learning. In Q-Learning, Q-function is used to predict the expected total reward. Q-function is updated based on Below equation learning rate α adjusts weight ratio of current q-value when q-value is updated. γ is discount rate which decides the importance of the rewards given in later steps.

$$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha(R(s_t,a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1}))$$

Deep Q-Network: DQN (Mnih et al., 2015) (Mnih et al., 2013) is an extension of Q-learning. DQN uses deep neural network to represent Q-function. DQN also uses experience replay. Experience replay is a method to use previous trajectory within the last fixed period for updating Q-function.

2.3 Offline RL

Traditional RL supposes interaction with environment to collect information. However, infinite times of interaction with environment is impossible in real application. For example, recommender system limits the times to interact with environment because the time of user access is not infinite. In autonomous driving, learning model from scratch is dangerous, because infant model will causes traffic accident. Social Game often changes the game setting like introducing new function to the game. It would be very costly to learn from scratch each time game change the setting slightly. Hence, the problems of RL to adopt real application are below.

- Problem 1 : the times to collect data or available data is limited.
- Problem 2 : pre-training model which provide minimum required ability is needed.
- Problem 3 : changing environment setting makes the model useless.

In Offline RL, agent learns from the fixed dataset. Offline RL tackle the problem 1 and 2. Offline RL doesn't always solve Problem 3. This research tackles problem 3. Suppose that dataset is a bit of datum which consists of state, action, transited state and reward. Offline RL uses this dataset to learn from. Learning from fixed dataset causes problem that the information agent can use is limited. In detail, agent don't uses the action sequences which is not contained in dataset. This force agent to learn the model to maximize their utility in the distribution dataset gives. The distribution which is not contained in dataset called out-of-distribution (OOD). It is important to keep utility in OOD having no-effect to utility of q-function. The rest of this section focuses on introduction of method for Offline RL.

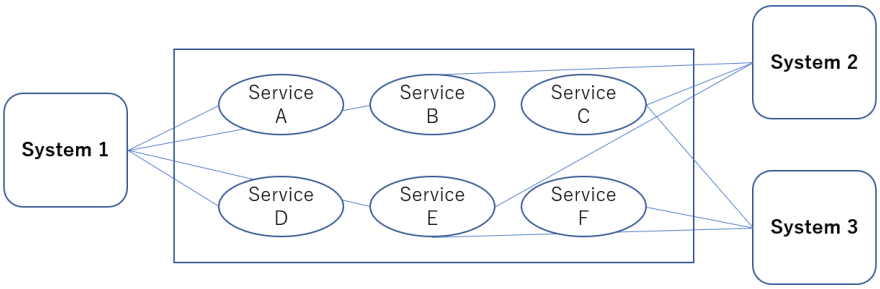

Figure 1: In SOA, multiple services are combined to build a system. It is also possible to divert the same service to different systems.

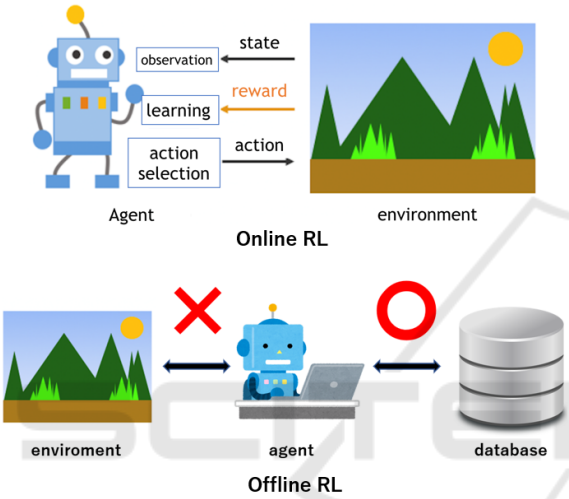

Figure 2: Explanation of online RL and offline RL.

Conservative Q-Learning: In Offline-RL or off-policy RL, overestimation of values induced by the distribution shift between the dataset and learned policy can cause failure of RL learning. In addition to this, Offline RL promise to learn effective policies from previously-collected, static datasets without further interaction. Conservative Q-Learning (CQL) (Kumar et al., 2020) aims to address these limitations by learning conservative Q-function such that the expected value of a policy under this Q-function lower-bounds its true value. CQL can be adopted to both of Q-Learning and actor-critic method. Actor critic is methods in which policy function and value function are learned explicitly. CQL can be combined with particular choice of regularizer. Below equation is instance of updates for CQL method with particular choice of regularizer $\mathcal{R}(\mu)$.

$$\min_{Q}\max_{\mu}\{\alpha(\mathbf{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)]-\mathbf{E}_{s\sim\mathcal{D},a\sim\hat{\pi}_{\beta}(a|s)}[Q(s,a)])$$
$$+\frac{1}{2}\mathbf{E}_{s,a\sim\mathcal{D}}[(Q(s,a)-\hat{\beta}^{\pi}\hat{Q}^{k}(s,a))^{2}]+\mathcal{R}(\mu)\}(CQL(\mathcal{R}))$$

3 RELATED WORK

3.1 Integrating Recurrent Neural Networks and Reinforcement Learning for Dynamic Service Composition (Wang et al., 2020)

This work adopts DQN and LSTM to capture time-series reward. Specifically, they use a recurrent neural network to predict the QoS, and then make dynamic service selection through reinforcement learning. This method focus only on the situation where the size of candidate services is very small.

3.2 A Deep Reinforcement Learning Approach for Large-scale Service Composition (Moustafa and Ito, 2018)

This work proposes an approach for adaptive service composition in dynamic and large-scale environments. The proposed approach employs deep reinforcement learning in order to address large-scale service environments with large number of service providers. This method uses DQN with Double DQN and prioritized experience replay. Their action space is from 100 to 200 and total service selection is from 600 to 800. This work uses three types of parameter to calculate QoS - availability, response time and reliability. They also consider the setting where part of services changes per concrete value episodes.

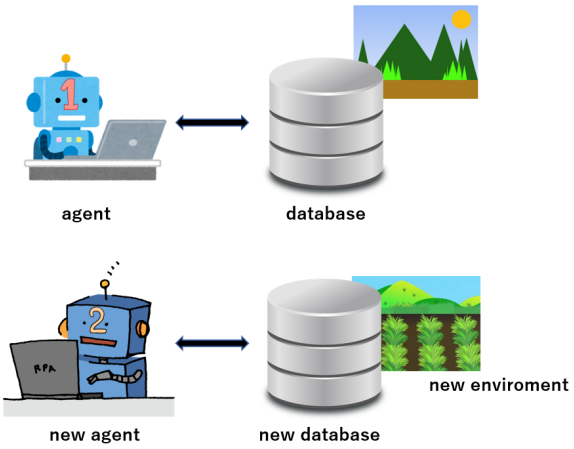

Figure 3: Re-learn with existing methods.

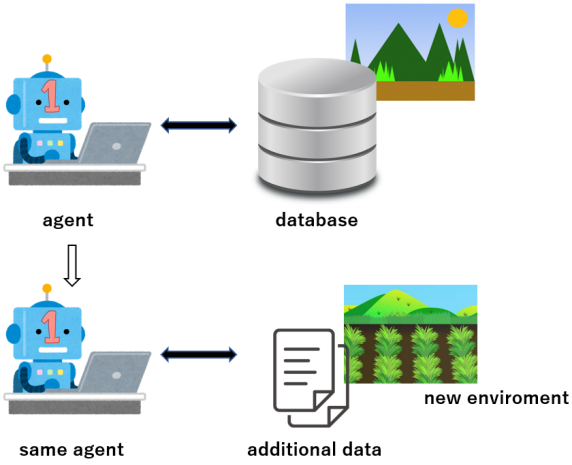

Figure 4: Re-learn with propose methods.

4 PROBLEM DESCRIPTION AND RL ARCHITECTURE

4.1 Objective

The purpose of the proposed offline reinforcement learning agent is to perform service selection for SOA system construction in a dynamic environment. The performance of the system depends on the combination of services. The agent learns to build a system with good performance. The premise is that the composition of services changes in SOA system building. Re-learning needs to be done efficiently when the composition of services changes.

With existing methods, new data is required every time the environment changes, and the learning process has to be redone from the beginning. The proposed method aims to improve retraining efficiency by inheriting models that have been trained in the environment before the change and prioritizing the sampling of additional data to be used during retraining.

4.2 Environment

Figure 5 shows the definition of MDP. In this research, WS-DREAM dataset (WSD,) is used to create environment which follows the definition of MDP. WS-DREAM is a Distributed REliability Assessment Mechanism for Web Services. WS-DREAM repository maintains 3 sets of data, QoS (Quality-of-Service) datasets ,log datasets, and review datasets. QoS datasets are used in this research.

Each state is assigned a service group consisting of n services. The agent selects one of the services from the service group by action. The reward is calculated based on the QoS datasets and normalized among the service group.

In this study, we experimented with the case where a service was added or removed from a group of services. When a group of services is changed, 10% of the services are removed and replaced with new ones. The method for relearning after the environment changes is described in the Replay Buffer section.

4.3 Model

Online RL Method: Online method is DQN based method. In this paper, the online method is used to prepare the replay buffer.

Offline RL Method: Offline Method expands online method for Offline Setting. This research adopts CQL method for offline setting and the method used in online setting is expanded by using CQL.

4.4 Replay Buffer

Offline reinforcement learning is trained using a fixed data set called replay buffer. In this study, the replay buffer is the history obtained from the training of the online method (DQN). We aim to improve the efficiency of relearning by selecting the data to use for the replay buffer. We select the data used for the replay buffer using the following algorithm.

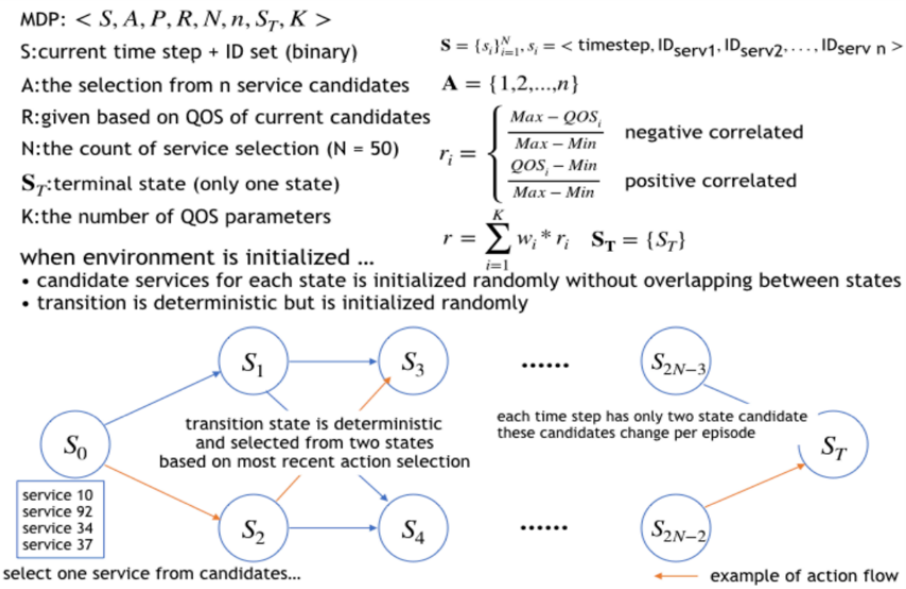

Figure 5: Definition of MDP.

Algorithm 1: Obtaining the replay buffer of the proposed method.

List of elements that have been added/removed L
Initialize replay buffer B to capacity N
while $N \geq$ size of B **do**
  $e \leftarrow$ An element chosen randomly from L
  $D \leftarrow$ Randomly generate an episode with e
  add D to B
end while

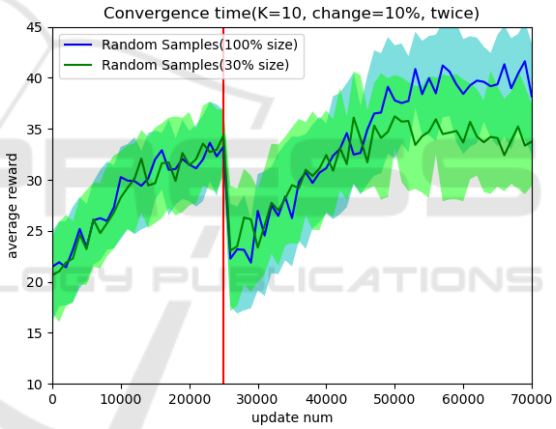

Figure 6: Reward transitions of 30% and 100%.

5 EXPERIMENTAL RESULTS

We use data of 50000 episodes obtained by executing DQN in the same environment for the first 25000 steps. After 25000 steps, we delete 10% of the existing services and replace them with new ones. Then the model is retrained.

5.1 Replay Buffer Size during Relearning

In this experiment, we investigate the replay buffer size required for retraining. Figure 6 compares reward transitions during retraining when the replay buffer data is prepared with the same size as the original one and with 30% of the original one. When the size of the replay buffer is 30%, the reward after relearning is low. This may be due to the fact that there is not enough data for relearning.

Figure 7 compares reward transitions during retraining when the replay buffer data is prepared with various size. In this experimental setting, the results were almost the same if the data size of the replay buffer used for relearning was at least 70% of the original. Depending on the magnitude of the environmental change, the data size required for relearning may not need to be as large as the initial training. If there is too little data, the converged reward tends to be small.

5.2 Importance Sampling of Data in Replay Buffer

In this experiment, we investigate the sample used for the replay buffer during relearning. Figure 8 compares reward transitions during retraining when replay buffer data is prepared randomly and when replay buffer is prepared with importance sampling. The replay buffer prepared with importance sampling con-

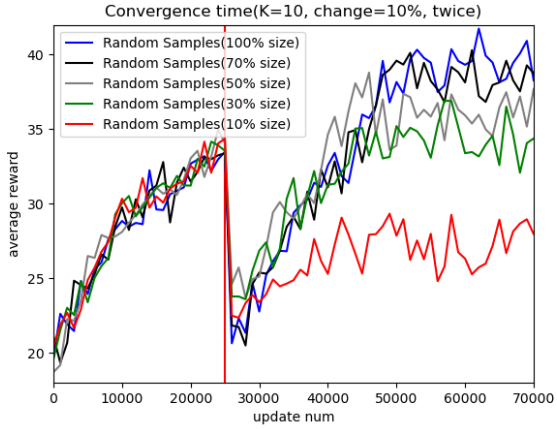

Figure 7: Reward transitions of 10%,30%,50%,70%, and 100%.

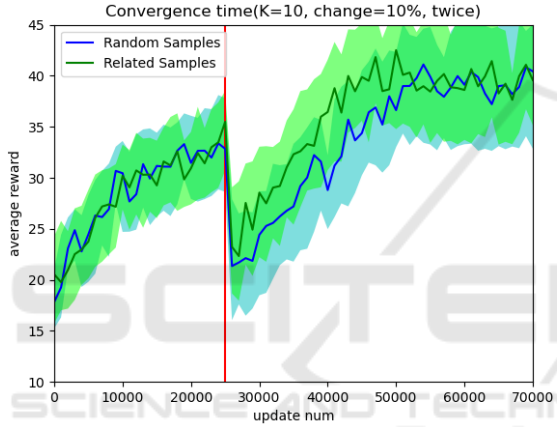

Figure 8: Reward transitions of 30% with propose method and 100% with random.

sists of data related to environment changes. In this experimental setting, convergence is faster when the replay buffer is prepared with the proposed method than when the replay buffer is prepared with random data.

6 DISCUSSION ON THE PROPOSED METHOD

The above experiments show the sample efficiency of the proposed method in relearning in dynamic environments. The proposed method is effective for practical applications of reinforcement learning in dynamic environments where sample acquisition is costly. Since the implementation is based on offline reinforcement learning methods, it is relatively easy to adapt the proposed method to existing systems. On the other hand, the definition of data related to changes in the environment varies greatly depending on the problem setting. It is not clear what factors should be prioritized when environmental changes are complex. Also, there is room for future verification of the degree to which the system can handle changes.

7 CONCLUSION AND FUTURE DIRECTIONS

In this study, we proposed an offline reinforcement learning method in a dynamic service selection environment in SOA. The proposed approach takes a model learned in the environment before it is modified, and performs efficient relearning by data related to the environment differences. The proposed method is able to adapt to changes in the environment with less data and shorter time.

The appropriate replay buffer size during relearning is treated as a hyperparameter. In the proposed method, all the data used in the replay buffer are related to the changes in the environment, but this ratio also needs to be considered.

ACKNOWLEDGEMENT

This work has been supported by Grant-in-Aid for Scientific Research [KAKENHI Young Researcher] Grant No. 20K19931.

REFERENCES

WS-DREAM Dataset. https://github.com/wsdream/wsdream-dataset.

Ie, E., Jain, V., Wang, J., Narvekar, S., Agarwal, R., Wu, R., Cheng, H.-T., Chandra, T., and Boutilier, C. (2019). Slateq: A tractable decomposition for reinforcement learning with recommendation sets.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Moustafa, A. and Ito, T. (2018). A deep reinforcement learning approach for large-scale service composition. In *PRIMA*.

Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*, pages 3–12. IEEE.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Wang, H., Li, J., Yu, Q., Hong, T., Yan, J., and Zhao, W. (2020). Integrating recurrent neural networks and reinforcement learning for dynamic service composition. *Future Generation Computer Systems*, 107:551–563.

Wang, S., Jia, D., and Weng, X. (2018). Deep reinforcement learning for autonomous driving. *arXiv preprint arXiv:1811.11329*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.