Anomaly-Aware Adaptation Approach for Self-Adaptive Cyber-Physical System of Systems Using Reinforcement Learning

Eunho Cho, Gwangoo Yeo, Eunkyoung Jee, and Doo-Hwan Bae
School of Computing
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon, Republic of Korea
ehcho@kaist.ac.kr, gwangoo525@kaist.ac.kr, ekjee@se.kaist.ac.kr, bae@se.kaist.ac.kr

2022 17th Annual System of Systems Engineering Conference (SOSE) | 978-1-6654-9623-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/SOSE55472.2022.9812671

***Abstract*—A cyber-physical system of systems (CPSoS) is a system composed of multiple constituent systems that interact with both physical and cyber environments. Self-adaptivity is essential for CPSoS because it works on both cyber and physical uncertainties in various environments. Main obstacles to achieving self-adaptive CPSoS are time constraints and system anomalies. An adaptation should be processed within a certain period and it should consider anomalies caused by system changes due to mechanical faults, cyber-attacks, or emergent behaviors. However, since existing adaptation approaches cannot fully handle both aspects, this paper proposes an advanced approach, A^4, for a self-adaptive system that can handle known anomalies in runtime. This approach learns the known anomalies before runtime and mitigates their impact when they are detected. We evaluated the A^4 approach for virtual and physical CPSoS and showed that A^4 was more efficient than other approaches.**

***Index Terms*—cyber-physical system, system of systems, self-adaptive system, system anomaly, reinforcement learning, transfer learning**

I. Introduction

A cyber-physical system of systems (CPSoS) comprises multiple constituent systems that interact with physical and cyber environments. With the introduction of modern software technologies, such as IoT, the concept of CPSoS became a reality for making connections between the physical and cyber environment in complex system of systems. However, those connections induce the system to face the various and complex environments that CPSoS must deal with [1]. Thus, without a proper strategy for mitigating the various uncertainties of those environments, it is almost impossible to design software that achieves system goal reliability and safety for every possible environment [2].

Self-adaptation is considered essential for reducing effort and improving the system reliability in CPSoS [3]. A self-adaptive system engineering is an approach that enables the behavior and structure to be reorganized to achieve system goals at runtime in response to changing environments. Various adaptation approaches exist for the self-adaptive systems, one of which uses model checking (MC) techniques to verify whether adaptation tactics meet system goals [4]. These approaches compose a formal model, for example, a Markov decision process, and verify the model with properties based on system goals. Another adaptation approach based on reinforcement learning (RL) makes adaptation decisions based on knowledge learned offline or online [5]. Rewards are configured according to system goals, and the RL model trains offline and online to understand the system's behavior and environment. Furthermore, based on learning about the system and environment, the model selects an appropriate adaptation tactic for a given state of the system and environment.

Unfortunately, there are two obstacles to achieving self-adaptation in CPSoS that existing approaches cannot fully handle. The first is time constraints. A CPS should adapt to an environmental change within a certain period. For example, the traffic light system that adapts the duration of the traffic light based on the number of cars has to adapt within a few seconds before the traffic light cycle begins. MC-based approaches involve high time or memory costs for verifying whole systems. For complex systems like CPSoS, the complete adaptation process may fail under limited time and memory constraints. Despite statistical MC costing less than probabilistic MC, it nevertheless takes a long time to verify tactics [4]. Another challenge for CPSoS is anomalies. An anomaly can be defined as an irregularly happening undesirable event caused by internal/external CPSoS factors. System changes often cause CPSoS anomalies due to mechanical failures, cyberattacks [6], or emergent behaviors, which can be discovered in a system-of-systems [7]. RL-based approaches are not be able to handle or hardly handle this problem.

We propose an anomaly-aware adaptation approach, A^4. The A^4 learns the environment and system based on the RL technology and prepares the anomaly-aware tactic planner based on the transfer learning method. We provide the application and evaluation of the proposed A^4 approach using two self-adaptive testbeds, a self-adaptive traffic light, and a self-adaptive smart warehouse. We evaluated A^4 and other adaptation approaches on cost efficiency, anomaly-aware effectiveness, and the relationship between the anomaly and performance. Moreover, it was shown that the A^4 was more efficient than other approaches.

The remainder of this paper is organized as follows. Section II describes related work on self-adaptive systems. Sec-

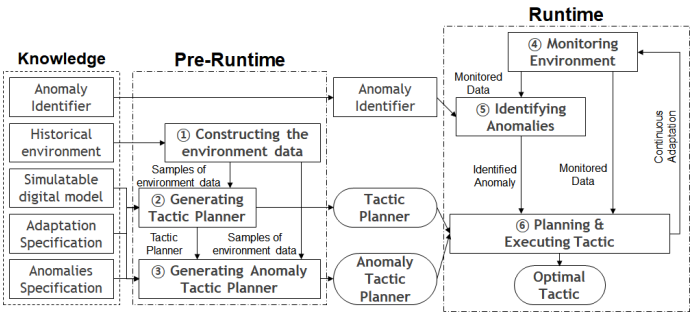


Fig. 1: Overall A^4 process

TABLE I: Specifications for an anomaly

	Principle (Anomaly)	**Example** Self-adaptive traffic light system (Car Accident)
Occurrence	How many components are occurring in the same anomaly? How many types does that anomaly have?	Four ways of the intersection
Behavior Change	How does the system change its behavior? How does the anomaly disrupt the adaptation goal?	Reduce the outflow of the intersection
Avg. Impact	How much does the anomaly affect the system?	Decrease the outflow by 50%
Avg. Duration	How long does the anomaly affect the system?	1 hour (360 tick)
Identified by	How can the system identify the anomaly?	By cameras, police report
Identified after	How long does it take for the system to identify an anomaly?	After 10 seconds (1 tick)

tion III provides details of the A^4, and Section IV explains the evaluation. Section V concludes the paper.

II. Related Work

There have been various existing adaptation approaches that are able to apply to self-adaptive CPSoS. Stevens et al. [8], and Shin et al. [4] improved and resolved the complexity of the MC technique and applied it to a self-adaptive system. Shin et al. [4] proposed an approach of using statistical model checking (SMC) instead of probabilistic model checking. Approaches that improve MC techniques can take advantage of costs. However, the evaluation of these approaches is limited to simple self-adaptive systems with a small number of tactics and no physical components.

Palm et al. [9] utilized the reinforcement learning (RL) technique to verify the adaptation tactics whether the tactic meets the system goal. Palm et al. [9] developed a policy-gradient-based online RL approach for a self-adaptive system to enable the reaction to the changing environment. RL-based approaches are common adaptation approaches because solving a problem with RL techniques resembles finding an optimal tactic in a self-adaptive system. However, ML models cannot handle drastic changes and have not yet been proven to converge on the optimum.

Quin et al. [10] and Cámara et al. [5] used both model checking and reinforcement learning techniques to ensure both time constraint and performance. Quin et al. [10] proposed ML techniques to find appropriate adaptation tactics that were likely to be optimal. The approaches resembled RL-based approaches but could only be used for systems with QoS goals and had the same limitations about anomalies.

III. Approach

Figure 1 shows the overall A^4 approach, which aims to select the optimal tactic for a given system state while mitigating the effects of potential CPS anomalies. The process is divided into pre-runtime and runtime phases. Before the process begins, A^4 requires knowledge of target anomalies and their specifications, anomaly identifiers, historical environmental data, simulatable digital models, adaptation tactics, and goal specifications. In the pre-runtime phase, a tactic planner is generated. (Step 1) The generation process requires samples of the environment based on historical environmental data. (Step 2) RL generates the tactic planner via a simulatable digital model and adaptation specifications. (Step 3) The generated tactic planner helps generate anomaly-specific tactic planners, which are the tactic planners for anomaly occurrences. By transferring learning, an anomaly tactic planner can easily be generated. (Step 4) During runtime, the system monitors the system and captures the environment. (Step 5) The anomaly identifier analyzes the system state and detects the anomaly. (Step 6) The generated tactic planner finds the optimal tactic for a given system state and identifies anomalies. Steps 4–6 are repeated throughout the complete system runtime.

Historical environmental data are used to learn the system and environment offline. The A^4 approach requires an executable system model, i.e., an abstraction of the target cyber-physical self-adaptive system. The model can be written in any modeling language, and it can contain any information that an engineer selects; however, it should be able to carry out step-by-step execution and extract the system state from logs. The simulatable model also contains a set of possible adaptation tactics, and each adaptation tactic can be executed by reconfiguring the system. The system should adapt to meet the adaptation goal, and the success can be measured by adaptation reward based on the system state.

Anomalies in the system are an essential part of the A^4 approach. A CPSoS anomaly is an irregularly happening undesirable event caused by internal/external CPSoS factors. A^4 can target and mitigate the negative impact of known CPSoS anomalies when an engineer defines the specifications, as shown in Table I. The engineer should specify the occurrence, behavior change, average impact, average duration, and anomaly identification. Occurrence refers to the types of anomalies. The same anomaly can occur in different components with similar impacts. Behavior changes and impacts reveal how the system changes due to the anomaly, which needs to be considered in a simulatable digital model. The requirement is the identification of anomalies.

(Step 1) A^4 constructs virtual environment data based on the historical environmental data. This data is used for the environment of the system's simulatable digital model. The environment data should be non-deterministic, meaning each

sample should have differences and uncertainties. If the data are always the same, the tactic planner can be overfitted to the fixed environment. Time-series forecasting techniques with a random initial value can help sample the environment, such as by using CNNs or RNNs [11] or injecting random values based on a random walk model [12]. Simple methods, such as injecting random values or selecting the value with normal distribution, can be effective solutions. The essential factor is that data should be realistic and non-deterministic. If sufficient data already exists, this step can be skipped.

(Step 2) A^4 generates the tactic planner using RL. Any RL technique can be used for the generation, such as the DQN [13] technique. Engineers can select the appropriate RL method based on the system's characteristics, adaptation tactics, adaptation goals, and anomalies. When A^4 generates the tactic planner, the simulatable system model performs the role of an RL simulation model. Based on the samples, the system model obtains tactics from the RL model, changes the system state, gives rewards, and returns the system state to the model. During this period, the tactic planner does not consider anomalies.

(Step 3) A^4 generates the anomaly tactic planner using transfer learning. For each anomaly that A^4 targets, an anomaly tactic planner should be generated. The first step is giving the anomaly information to the simulation. These anomalies should be specified, and the simulatable digital model should be able to simulate each anomaly. Then the anomaly-free tactic planner is imported to initiate the transfer learning. With the anomaly-free tactic planner, the model retrains step-by-step based on the simulation results of the specific anomaly. In this step, the simulation returns the results for the anomaly, which remains present throughout the training. Please refer to the thesis [14] for the detailed algorithm for generating the anomaly-specific tactic planner.

(Step 4) The system is on runtime. It monitors the environment continuously and detects the need for adaptation. The environment can be monitored by sensors connected to the physical parts of the system. The system state is also monitored. (Step 5) The system detects the anomaly and identifies it. There are various methods of anomaly identification. Additional sensors for detecting anomalies or CPS anomaly detection algorithms can be used for physical anomalies such as faults due to wear-out. Exceptions, assertions, and cyberattack protection algorithms are examples of anomaly identifications for cyber components.

(Step 6) Based on the monitored environment and system state, the A^4 approach decides whether an adaptation is required or not. For general cases with no anomalies, the anomaly-free tactic planner plans the adaptation tactic and finds the optimal tactic for the system. However, if there is an identified anomaly, the tactic planner is changed to an anomaly-specific tactic planner trained on the digital model for a particular anomaly. The selected tactic planner then finds the optimal tactic based on the given environment and system state and executes the selected tactic. Please refer to the thesis [14] for the detailed algorithm.

TABLE II: Descriptions of self-adaptive systems

Testbed	**Traffic Light (SATLS)**	**Smart Warehouse (SASWS)**
Environment	Car inflow to the target intersection	Received orders into the warehouse
Anomaly	Car accident	Jammed item
Occurrence	Four routes	Two repository devices
Behavior Change	Reduce the outflow of the intersection	Slows the item flow
Avg. Impact	Decrease the outflow by 50%	Item moves at 25% of normal speed
Avg. duration	360 ticks (1h) It does not reoccur until 2h after it ends.	10 ticks (physical) 100 ticks (simulation)
Identified by	Sensors (cameras) and police reports	Sensors on the repository device
Identified after	After one adaptation cycle	After 1 tick passed
Experiment Ends	After 8,640 ticks (24h)	After all the orders are completed
Rewards	Number of cars waiting	Order complete: 30 Item removed: -70 Order waiting: -1
Anomaly constant	**108,000**	**5** for Physical **30,000,000** for Virtual
E(**time b/w anomalies**)	1,426.82	2.87 for Physical 6,838.24 for Virtual
E(**# of anomalies**)	3.45	1.55 for Physical 1.44 for Virtual

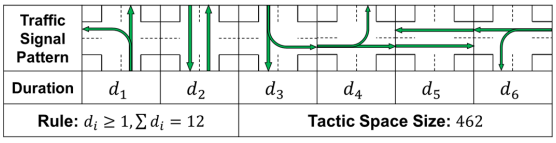


Fig. 2: Traffic signal pattern for the SATLS

IV. Evaluation

Three research questions were developed based on the motivation and goals:

- RQ1: *Cost Efficiency*. How does A^4 handle adaptation within time constraints?
- RQ2: *Adaptation Effectiveness*. How does A^4 handle anomalies compared to other approaches?
- RQ3: *Relationship between Anomaly and Performance*. How does the adaptation effectiveness of A^4 change for various anomaly occurrence probabilities?

A. Target Self-Adaptive Systems and Anomalies

Two test beds were used for the evaluation, both of which represented examples of cyber-physical systems. Table II gives their descriptions and experiment setups.

1) Self-adaptive Traffic Light System (SATLS): A SATLS is an excellent example of a self-adaptive CPSoS that needs to be aware of anomalies such as traffic accidents. The flow of cars changes over time. There may be traffic congestion during rush hour, and sometimes, there may be a traffic accident that blocks a route and causes more traffic congestion. According to historical data, a SATLS predicts the number of cars. Based on predictions of the future environment, the signal controller finds the optimal configuration for the traffic light duration that minimizes the number of waiting cars at an intersection.

This study uses a SATLS as a virtual testbed, which was implemented in [4]. Figure 2 shows the traffic signal pattern of the system. One traffic signal cycle comprises 6 component

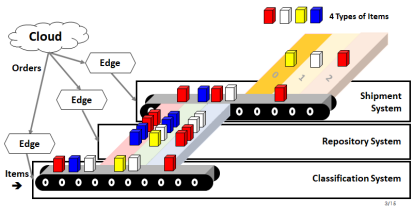


Fig. 3: Overview of the SASWS

patterns, each of which is assumed to have at least one tick (in this example, 10s), and the sum of the durations for all components is assumed to be within 12 ticks (2 mins). In this configuration, the number of tactics is 462 ($\binom{11}{5}$). For this example, the tactic is decided based on the environment at the start of the cycle, and the decision should be made before the first pattern ends. Thus, a reasonable time constraint was 10s, which is the minimum time of the first pattern.

2) Self-adaptive Smart Warehouse System (SASWS): We implemented virtual and physical SASWS to evaluate A^4. It employs a self-adaptive edge-computing-based system to reduce the time required to process orders. Figure 3 shows an overview of such a system. The warehouse contains four types of items. The classification subsystem classifies the items based on their type and sends them to the repository subsystem. The repository subsystem holds the items and releases them when orders are received. The shipment subsystem sends each item to its destination.

Each subsystem consists of one edge server and one to three LEGO EV3 devices. Each LEGO EV3 device controls a conveyor belt and provides temporary storage for items. The device communicates with the edge server (the raspberry pi) and follows the direction of the edge server. The edge servers collect the system state from devices and other edge servers and decide based on predefined rules. The decision-making of the edge server depends on the direction of the cloud server. A virtual customer sends an order to the cloud server, then examines the environment based on data from the edge servers and selects adaptations. Each communication is based on the HTTP API post that the edge or cloud server provides. The edge and cloud servers were created using the Python Django REST framework to respond to communications and log messages and data efficiently.

Adaptation in the SASWS was based on the environment and orders. Orders were generated based on the rule of uncertainty. The SASWS could adapt the repository for each item type; for example, if there are three repositories (e.g., r0, r1, and r2, respectively), items 0 through 3 would store i0, i1, i2, and i2 for each. The item can be stopped in the middle of the conveyor belt due to the uncertainty of the motors. This 'stuck item' is assumed as an anomaly of this system. Each item could pass through the repository subsystem 50s after the blockage occurred. In this research, only two types of stuck items were considered target anomalies and stuck on two side repository devices.

Because the overall reliability of the SASWS's physical components was inadequate for evaluating substantial amounts of orders and items; therefore, the adaptation tactic was changed to the single-item level. Also, the anomaly was artificially generated in the system to provide a fair comparison. Moreover, the SASWS functions were based on discrete ticks, meaning that in one tick, the classification subsystem and shipment subsystem could only serve one item, and the repository subsystem could only release one item per device.

3) Anomaly Generation: Because the comparison had to be fair, this research generated the anomaly artificially. The cause of the CPS anomaly differs, and the anomaly's distribution also varies. In this study, the anomaly was assumed to follow the wear-out failure rate: $p = 1 - e^{-t/C}$. p shows the probability of an anomaly when the time after the anomaly's occurrence is t, and C shows the value that controls the distribution of the anomaly. Both testbeds used the approximated equation to artificially create an anomaly based on the given probability. The SATLS had approximately three or four anomalies for each simulation to show the anomaly at various times, such as dawn, morning, afternoon, or night. The SASWS had at least one anomaly for each experiment.

B. Experimental Design

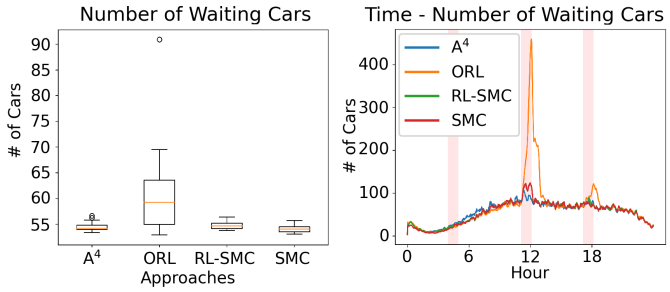
An SATLS was used for this experiment of RQ1 - cost efficiency. The same environment (car inflow) and anomaly (traffic accident) for each adaptation approach evaluation were given. The system ran for 24h, and it adapts the traffic light duration for every cycle (120s). The system pauses during the adaptation planning to measure the adaptation planning time. The time before and after adaptation was measured, and the sum of all adaptation planning times constituted the result. Each approach experiment was repeated 20 times.

Both SATLS and SASWS were for the experiment of RQ2 - anomaly-aware effectiveness. The experimental process for the SATLS was the same as that mentioned in the experimental design of RQ1. However, this experiment measured the rewards for the number of cars waiting at the intersection.

The experiments for the SASWS were based on digital model simulation and the physical testbed. The same environment and anomalies of the SASWS were utilized between the approaches. Because the time constraint of the system is short, the RL-based approach and random approach were only used for comparison. The experiments ended when the system completed a fixed number of orders. Predefined rewards and the times until the experiments ended served as results. The experiments were repeated 20 times for each approach.

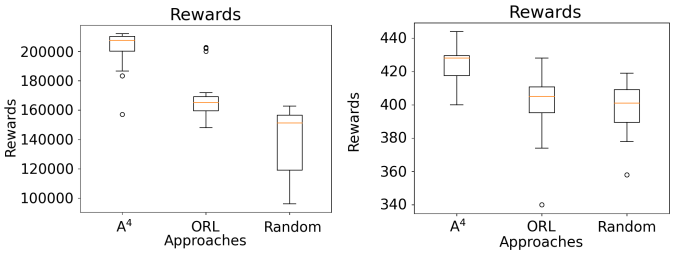
The experiment for RQ3 - the relationship between anomaly and performance concerned the SATLS. Also, only the A^4 and ORL approaches were evaluated. Other experimental designs were the same as those in the design of RQ1 and 2, but the number of anomalies changed. For each approach and anomaly constant, the simulation was run 1,000 times.

We also employed SMC [4], online RL (ORL) [9], and RL with the SMC method (RL-SMC), inspired by the study in [10], to compare them with the A^4 approach. The RL-SMC method verifies and selects the tactic that only passes certain



(a) Box plot for the avg. cars (b) Example result

Fig. 4: Evaluation result of SATLS



(a) Simulated SASWS (b) Physical SASWS

Fig. 5: A box plot result for the reward for the SASWS

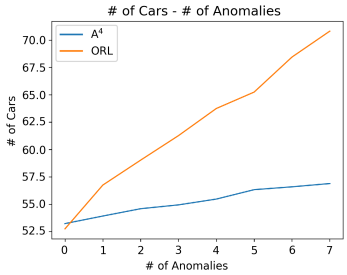


Fig. 6: Number of cars changes based on the number of anomalies

criteria by the RL model. The SMC verified the tactic and made the best choice. For the SATLS, all four approaches were approaches to compare, and for the SASWS, only the RL-based, A^4, ORL, and random approaches were the approaches to compare. The A^4 approach and ORL approach used the same CNN model with the deep Q-learning algorithm [13] and the same hyperparameters to train the model. The pretraining process was also the same as the ORL approach; therefore, the only difference between the ORL model and the A4 anomaly-free tactic planner was the online learning process.

C. Results and Analysis

1) RQ1. Cost Efficiency: A^4, ORL [9], RL-SMC [10], and SMC [4] reported 0.584ms, 1.887ms, 25.49s, and 40.30s for average adaptation time, respectively. The results showed that the A^4 and ORL approaches took less than 1s for the single adaptation process, while other MC-based approaches took more than 10s. The RL-SMC approach successfully reduced the adaptation size, but it took more than 20s to adapt, violating the time constraints for this testbed. The ORL approach took slightly longer than the A^4 approach because of the online learning process but did not violate the time constraint. A^4 gave the best results among the approaches.

2) RQ2. Anomaly-Aware Effectiveness: Figure 4a shows the box plot result for the effectiveness of the SATLS. All four approaches had 50 to 60 cars waiting on average for a 24h simulation. In particular, the A^4, RL-SMC, and SMC approaches had comparable results for average reward values and stable results for every run. Although the RL-based and A^4 approaches have not yet proven convergence, based on the results for the SATLS, it can be argued that the proposed approach is competitive with other approaches. The ORL approach had approximately 60 cars waiting on average, but it showed a large dispersion and recorded the highest value. The differences between the other approaches and the ORL approach were not significant and were mainly caused by anomalies.

Figure 4b shows a one-run example result for the RQ2 experiment. The red zone shows that the anomalies occurred at particular times—in this example, three anomalies occurred at almost 4 a.m., 11 a.m., and 5 p.m. The anomalies did not affect the performance at 4 a.m. when a few cars passed, but they affected the performance of the ORL approach at 11 a.m. and 5 p.m. The online RL algorithm was good for continuously adapting the model to the changing system and environment; however, it required time to collect sufficient data to change. In this experiment, the RL algorithm could not collect enough data for online learning (only 30 min). Moreover, the online RL algorithm had another disadvantage when the anomaly was resolved—the system changed once more, and the extensive memory of anomalies damaged the online RL algorithm.

Figure 5a show box plots of the result for the effectiveness of the simulated SASWS. The performance of the A^4 approach exceeded that of the ORL and random approaches. The ORL approach had higher rewards than the random approach and recorded similar values on some runs; however, the average result was much lower than that of the A^4 approach. Nevertheless, the A^4 approach shows better performance than other results.

Based on the results of the simulation sessions, the experiment was conducted in an physical SASWS. Figure 5b shows the box plot result for the effectiveness of the physical SASWS. Because the size of the experiment was smaller than the simulation, the differences between the approaches were somewhat unclear; however, the A^4 constantly recorded high rewards and short processing times based on the box plot results, meaning that the A^4 approach was influential not only in the simulation but also in the real world.

3) RQ3. Relationship between Anomaly and Performance: Figure 6 also shows the change in the number of cars based on the anomaly. The average number of cars increased for both approaches as the anomaly frequency increased. The ORL approach had superior results when no anomaly was present, but the A^4 approach showed that the increased values were not

as high as the ORL approach. The A^4 approach showed only a four-car increase compared to the ORL approach's increase of almost 20 cars for seven anomalies. This result showed that the A^4 approach was better than the ORL approach for managing anomalies and mitigating performance degradation. Moreover, the A^4 approach was able to manage situations regardless of the number of anomalies.

D. Discussion

One threat to validity was the selection of the A^4 RL model. This research was based on a deep Q-learning algorithm, action–reward function-based, supporting discrete tactic space. This algorithm has the limitation of not supporting continuous tactic space, but this threat was mitigated because many self-adaptive systems have discrete sets of adaptation tactics. Also, other RL algorithms could be used instead of deep Q-learning. Another threat was the fair comparison between the A^4 and baseline approaches, such as online RL or SMC. In this research, the implementation of baseline approaches followed the step-by-step guide of approaches of other studies [4], [9], [10]. Moreover, the complexity, environment, anomalies, and pre-trained RL models were equally given as the baseline approaches.

The experiment was conducted on a small CPSoS, and in this research, an single SATLS and an SASWS were the experimental subjects. However, the A^4 approach can be applied to any CPSoS with known anomalies. The subjects were selected to compare approaches using a large number of experiments; therefore, it was confident that the A^4 approach would show similar experimental results to this evaluation.

One limitation of the research is the difficulty of the A^4 application. This approach requires diverse types of knowledge, such as anomaly specifications and anomaly identifiers. It is almost impossible to know all the anomalies that may affect the system. Moreover, anomalies in the field have been actively researched, but appropriate technologies have not yet been developed. It is also challenging to develop a digital model that can imitate anomalies. Another limitation is the inefficiency of anomaly-specific tactic planners. Making a separate tactic planner is relatively easy when using transfer learning, but if there are many anomalies and an engineer considers a situation in which multiple anomalies co-occur, the number of anomaly-specific tactic planners grows exponentially alongside the state explosion problem.

V. Conclusion

CPSoS faces various changing and evolving environments. Self-adaptation is the key to achieving the CPSoS goal by adapting systems' behavior to changing environments. This paper proposed A^4, an anomaly-aware adaptation approach to solve both time and anomaly problems. A^4 generates anomaly-specific tactic planners using transfer learning and plans adaptation when anomalies occur. A^4 was shown fast enough not to violate any testbed time constraints, and its adaptation performance for various anomalies was proven using physical testbeds. Future research can aim to solve the limitations of tactic planners by generating an integrated anomaly tactic planner and considering an integrated approach using an anomaly identifier based on deep learning.

Acknowledgment

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2022-2020-0-01795) and (No. 2015-0-00250, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System) supervised by the IITP(Institute of Information & Communications Technology Planning & Evaluation).

References

[1] F. D. Macías-Escrivá, R. Haber, R. Del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," Expert Systems with Applications, vol. 40, no. 18, pp. 7267–7279, 2013.

[2] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel et al., "Software engineering for self-adaptive systems: A second research roadmap," in Software Engineering for Self-Adaptive Systems II. Springer, 2013, pp. 1–32.

[3] M. D'Angelo, A. Napolitano, and M. Caporuscio, "Cyphef: a model-driven engineering framework for self-adaptive cyber-physical systems," in Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, 2018, pp. 101–104.

[4] Y.-J. Shin, E. Cho, and D.-H. Bae, "Pasta: An efficient proactive adaptation approach based on statistical model checking for self-adaptive systems," Fundamental Approaches to Software Engineering, vol. 12649, p. 292, 2021.

[5] J. Cámara, H. Muccini, and K. Vaidhyanathan, "Quantitative verification-aided machine learning: A tandem approach for architecting self-adaptive iot systems," in 2020 IEEE International Conference on Software Architecture (ICSA). IEEE, 2020, pp. 11–22.

[6] J. Plasse, J. Noble, and K. Myers, "An adaptive modeling framework for bivariate data streams with applications to change detection in cyber-physical systems," in 2017 IEEE International Conference on Data Mining Workshops (ICDMW). IEEE, 2017, pp. 1074–1081.

[7] H. Kopetz, A. Bondavalli, F. Brancati, B. Frömel, O. Höftberger, and S. Iacob, "Emergence in cyber-physical systems-of-systems (cpsoss)," in Cyber-physical systems of systems. Springer, 2016, pp. 73–96.

[8] C. Stevens and H. Bagheri, "Reducing run-time adaptation space via analysis of possible utility bounds," in 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, 2020, pp. 1522–1534.

[9] A. Palm, A. Metzger, and K. Pohl, "Online reinforcement learning for self-adaptive information systems," in International Conference on Advanced Information Systems Engineering. Springer, 2020, pp. 169–184.

[10] F. Quin, D. Weyns, T. Bamelis, S. S. Buttar, and S. Michiels, "Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning," in 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE, 2019, pp. 1–12.

[11] B. Lim and S. Zohren, "Time-series forecasting with deep learning: a survey," Philosophical Transactions of the Royal Society A, vol. 379, no. 2194, p. 20200209, 2021.

[12] F. Spitzer, Principles of random walk. Springer Science & Business Media, 2013, vol. 34.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.

[14] E. Cho, "Anomaly-aware adaptation approach for self-adaptive cyber-physical system of systems using reinforcement learning," Master's thesis, Korea Advanced Institute of Science and Technology (KAIST), 2022.