Learning-Based Self-Adaptive Assurance of Timing Properties in a Real-Time Embedded System

Mahshid Helali Moghadam[1,3], Mehrdad Saadatmand[1], Markus Borg[2], Markus Bohlin[1], Björn Lisper[3]
[1]RISE SICS Västerås, Sweden
[2]RISE SICS Lund, Sweden
[3]Mälardalen University, Västerås, Sweden
mahshid.helali.moghadam@ri.se, mehrdad.saadatmand@ri.se, markus.borg@ri.se, markus.bohlin@ri.se, bjorn.lisper@mdh.se

***Abstract*—Providing an adaptive runtime assurance technique to meet the performance requirements of a real-time system without the need for a precise model could be a challenge. Adaptive performance assurance based on monitoring the status of timing properties can bring more robustness to the underlying platform. At the same time, the results or the achieved policy of this adaptive procedure could be used as feedback to update the initial model, and consequently for producing proper test cases. Reinforcement-learning has been considered as a promising adaptive technique for assuring the satisfaction of the performance properties of software-intensive systems in recent years. In this work-in-progress paper, we propose an adaptive runtime timing assurance procedure based on reinforcement learning to satisfy the performance requirements in terms of response time. The timing control problem is formulated as a Markov Decision Process and the details of applying the proposed learning-based timing assurance technique are described.**

Keywords—Timing properties; self-adaptive performance assurance; real-time embedded systems; reinforcement learning

I. INTRODUCTION

Nowadays, many industrial control systems are real-time programs mainly implemented on Programmable Logic Controllers (PLCs). In real-time control systems, performance-related requirements such as timing requirements for response time is a highly important aspect required to be analyzed, tested and guaranteed more rigorously than other types of software systems. One of the basic language elements of the PLC-based industrial control programs are function blocks. According to the programming languages standards of programmable controllers, a control program may consist of zero or more function blocks. Many of the main types of function blocks mostly work based on timers, therefore, timing requirements in terms of response time or time-out are essential parts of the performance-related requirements in real-time control programs.

Timer-based function blocks provide their outputs according to programmable/controllable time intervals. Correctness of the behavior of a complex real-time program highly depends on satisfying the timing requirements. For example, untimely behavior of the blocks might lead to total failure in the function of the program. Failure in meeting the timing requirements due to common reasons such as unexpected delay in timer-based function blocks could happen frequently.

The real-time control systems specifically in safety-critical applications, should be robust to some extent against deviations in the temporal behavior of the constituent components. In general, robustness has been defined as to which degree a system can work properly in the presence of incorrect inputs or stressed critical conditions [1]. In a PLC perspective, a robust real-time control program would have the ability to cope with trivial untimely behavior of the constituent function blocks. A wide variety of issues can affect the temporal behavior of a function block. Several approaches to preserve, guarantee, verify and model the timing constraints have been proposed based on monitoring timing properties at runtime. These approaches include using an extra scheduler module to try to preserve and guarantee the timing-related properties like execution time, inter-arrival time, deadline misses at run time [2, 3], generating the model of timing constraints based on monitoring results [4], predicting violations of timing requirements based on monitoring the temporal behaviors of real-time systems [5] and dynamic/runtime verification of timing properties using a test execution environment based on timed-automata model of the system [6].

In this paper, we propose a self-adaptive learning-based approach for response time assurance of a real-time control program. we present the procedural form of the proposed approach in an industrial real-time control program. This approach formulates the control process as a Markov Decision Process (MDP). It exploits a Reinforcement Learning (RL) algorithm, i.e. Q-Learning for adaptive control of response time to meet the performance target and preserve the performance property of the system against any deviations in the temporal behavior of the program components.

The rest of this paper is organized as follows; Section 2 discusses the motivation and background concepts of RL, Section 3 presents the details of the proposed approach, a short discussion on applicability of the approach and tuning its performance. Section 4 provides a summary of the related work and background approaches. Section 5 presents a conclusion. and future directions of this work-in-progress research.

II. MOTIVATION AND BACKGROUND

Performance assurance is preserving the performance properties of the system with respect to the changeable conditions. Providing performance assurance techniques for meeting the performance targets in terms of performance parameters like response time is essential for real-time systems. Model-driven performance assurance techniques require precise knowledge of the system, i.e. the system model. Adaptive learning-based technique helps meet the performance target of

the system with respect to the uncertainty in timely behaviors of the constituent function blocks, and without need to have a precise model of the system. Reinforcement learning-based techniques have been frequently used for preserving the non-functional properties like performance during runtime. For example, performance control of services in virtualized environments according to Service Level Agreement (SLA) is a common example. Several adaptive learning-based approaches [7,8] for guarantees of computation or memory resources have been proposed in this area. On the other hand, for real-time control programs in which the resources are fixed, it might also be essential to provide runtime assurance of performance-related behaviors of the constituent elements without having a precise model of the system.

A. Reinforcement Learning

A Markov Process [9] is a random process with a sequence of Markov states, where the next state of the system is independent of the past history of the current state (the Markovian property). A Markov Decision Process (MDP) is a Markov Reward Process along with decisions on states. It is described as a tuple <S, A, P, R, γ> where

- S is the set of states
- A is the set of actions
- P is the probability transition between states based on applied actions,
  $$P_{s\to s',a} = P\left[S_{t_n} = s' \middle| S_{t_{n-1}} = s, A_{t_{n-1}} = a\right] \quad (1)$$
- R is the reward function,
  $$R_{s,a} = E\left[R_{t_n} \middle| S_{t_{n-1}} = s,\ A_{t_{n-1}} = a\right] \quad (2)$$
- γ is a discount factor, $\gamma \in [0,1]$

MDP is a formal mathematical model to describe the decision-making process in a stochastic environment, particularly in learning problems. Reinforcement learning [10] is a type of learning that acts based on interaction with the environment. The learner agent observes the environment, takes an action and receives a reward signal based on the effects of the applied action. The agent uses a policy to select the actions in a way to maximize the long-term received reward. This policy could be a tradeoff between selecting the actions with high value based on the experiences (exploitation) and an exploration-based action selection. Finding an optimal policy to maximize the received reward through interaction with the environment is the objective of learning in an MDP.

Q-learning [10] is a well-known Temporal Difference (TD) algorithm in the scope of RL which learns the value function of the cumulative reward to find the optimal policy. It is an off-policy learning algorithm as the agent learns the optimal policy regardless of the action selection strategy.

III. Learning-based Assurance of Timing Properties in Real-time Embedded Systems

This section presents the details of the proposed learning-based approach for guaranteeing the satisfaction of response time as a performance property in a real-time program. We describe the details of the approach applied to a sample program running on an embedded system like a Programmable Logic Controller (PLC). According to the proposed software architecture for programmable controller systems in IEC 61131-3 [11], a program may consist of multiple function blocks (zero or more) and basic functions like AND, OR and NOT. Figure 1 shows the structure of the sample program in Function Block Diagram (FBD) format as an integration of some basic functions like AND, and a number of timer-based blocks like FLTDLY. Function Block Diagram (FBD) is a well-known standardized programming language for programmable controller systems.

FLTDLY is a function block working based on timer, that activates its output after a preset time interval (the *fault delay, FD*) following the activation of its own input. The function blocks are executed in a predefined execution order which is defined according to the rules in IEC 61131-3. The execution order satisfies the data dependency flow between function blocks. A control thread is responsible for preserving the execution order. The execution steps of each function block include reading the inputs, execution/computation of the function block and updating the output ports. Time deviations in the response time of timer-based function blocks can frequently occur due to various reasons, therefore, it may cause exceeding of the predefined response time of the program. The proposed approach tunes the response time of function blocks using a model-free learning-based technique to guarantee the satisfaction of response time (end-to-end execution time) threshold.

The main steps of the proposed RL-based technique are as follows:

1) *States.* The control scan thread is responsible for detecting the state of the program. After executing each function block, the control thread measures the state of the program in terms of *cumulative execution time. The cumulative execution time* measures the amount of time that has passed until the current execution point. The execution time until the end of n^{th} function block execution is defined as follows:

$$ET_n = \sum_{i=1}^{n} ET_i^f \quad (3)$$

where the ET_i^f is the execution time (response time) of the i^{th} function block. The cumulative execution time is classified into four classes according to the compliance with a predefined performance threshold. The cumulative performance threshold of the function blocks until the end of n^{th} function block, is defined as follows:

$$PT_n = \sum_{i=1}^{n} PT_i^f \quad (4)$$

Where PT_i^f is the predefined response time threshold of the i^{th} function block. The response time is formed by the sum of delay time and processing time of the function block. The states of the program could be described as a set of *acceptable, Low-Deviation, Medium-Deviation and High-Deviation* states, as shown in Fig. 2. In this paper, *PT* represents the performance threshold and also a tolerance region, [PT, PT'], which $PT' = PT+\varepsilon$, has been considered as an acceptable tolerance range.

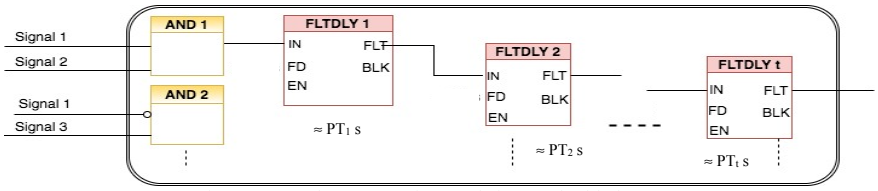

Fig. 1 A sample program in a real-time embedded system

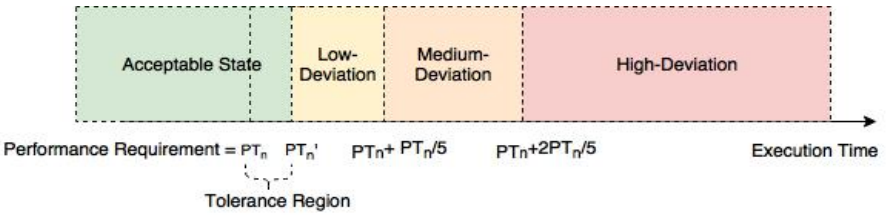

Fig. 2 States representation in the program

2) *Actions* are control operations to tune the *fault delay*, *FD*, parameter of the next running function block. A value as a minimum required delay, FD_m, has been considered for providing a safety margin for the execution of function blocks. The *FD* parameter of the blocks cannot be set to a value less than FD_m. Tuning actions are defined as follows:

$$TD_i = FD_i - FD_m \quad (5)$$

Where *TD* is the time distance between the preset fault delay of the i^{th} function block, FD_i, and the minimum required delay. Actions are decreasing the FD_i based on a factor in the following set:

$$DecFac = \left\{0, \frac{1}{3}, \frac{2}{3}, 1\right\} \quad (6)$$

Therefore, the possible actions are defined as follows:

$$Actions = \left\{noAction, \left\lfloor FD_i - \frac{TD_i}{3} \right\rfloor, \left\lfloor FD_i - \frac{2TD_i}{3} \right\rfloor, FD_i - TD_i\right\} \quad (7)$$

3) *Reward* signal is computed through the utility function defined as follows:

$$R(t) = \begin{cases} \frac{1}{\frac{PT_n}{10}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{ET_n - PT_n}{PT_n/10}\right)^2}, & PT_n < ET_n \\ 1, & ET_n \leq PT_n \end{cases} \quad (8)$$

The reward function is based on the probability density function of a Gaussian distribution with $\mu = PT_n$ and $\sigma = PT_n/10$. It provides a smooth trend in the range (0,1] for the reward values.

A policy defines a mapping between states and actions, and determines which action is taken in a given state. The expected return of taking action *a,* in state *s,* based on following the policy π, is stated by an action-value function $Q^{\pi}(s, a)$ defined as follows [10]:

$$Q^{\pi}(s, a) = E^{\pi}[R_t | S_t = s, A_t = a],$$
$$R_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^k r_{t+k+1} \quad (9)$$

The q-values as the experience of the quality control agent are stored in a look-up table, *Q-table*. The agent uses $Q(s, a)$ value as a utility value to decide the actions when it needs to take actions. The final objective of the learning procedure is to find an optimal policy to maximize the long-term reward. The optimal policy provides the optimal action-value function for pairs (s, a).

Tuning learning performance. Various action selection strategies could be exploited. The agent can select the actions randomly or based on the stored utility values in the Q-table. *ε-greedy* is one of popular strategies to make a trade-off between exploitation and exploration aspects of the action selection. The procedure of the learning-based timing properties assurance technique for a real-time control program is presented in List 1.

Applicability. Real-time control programs running on embedded systems are essential parts of many mission-critical industrial systems such as automotive control systems. The proposed approach could provide an adaptive runtime performance assurance in terms of end-to-end execution time for time-critical programs in embedded systems. The proposed approach finds the optimal policy of tuning delay parameters to guarantee the performance threshold of the program using an RL-based mechanism. The learning-based method will converge after a number of learning cycles. Regarding the tolerance degree in soft and hard real-time systems, the proposed approach in its incremental learning fashion can be used in a soft real-time system. Meanwhile, the performance of learning could be adjusted by tuning the learning parameters and action selection strategy. Regarding the characteristics of the hard real-time systems, since the system cannot tolerate the delayed result, the agent with the converged Q-table can be used as an assistant timing properties assurance technique. In general, this approach may be added as an extension to features of the scheduler or the execution supervision (control) program. It could provide a flexible capability of performance assurance in terms of response time for real-time control programs in embedded systems.

IV. Related Work

Most of the embedded systems run real-time programs. Some of the common distinguishing characteristics of a real-time system are timing properties, event-driven processing, multi-tasking, and runtime task scheduling. Timing properties are main features of real-time programs, primarily implying a specified limit on the response time or execution time of the system. In general, non-functional properties (NFPs), including timing properties, could be evaluated or verified against the non-functional requirements (NFRs) in a real-time system through the following approaches:

Algorithm: RL-based assurance of timing properties in a PLC-based real-time program

Required: S, A, Set the learning rate α and the discount factor γ
1. Initialize q-values, $Q(s,a) \forall s \in S, \forall a \in A$, arbitrarily
Repeat for the execution cycles of function blocks
2. Detect the current state of the program, s_n
3. Select a possible action for the current state using the action selection policy
(e.g. ε-greedy, select $a_n = argmax_{a \in A} Q(s_n, a)$ with probability 1- ε or a random action with probability ε)
4. Take the selected action, let the system run
5. Detect the state of the system at the end of next cycle and observe the signal reward.
6. Update the q-value by
$$Q(s_n, a_n) = Q(s_n, a_n) + \alpha[r_{n+1} + \gamma \max_a Q(s_{n+1}, a) - Q(s_n, a_n)]$$

1) Measurement-based analysis. This approach works through running the system in realistic environments to verify the properties. It suffers from long evaluation time and a lack of complete coverage of all possible system scenarios, particularly in large systems. Verification scenarios can be generated through different methods and run inside the test execution environments for real-time systems like Farkle [6].
2) Analytical Models. This approach performs a static analysis based on an analytical model of the system. This analysis approach provides a processing model for the system and generates the output model regarding the input, and resource models. However, analytical analysis mostly considers assumptions that partially limit the efficiency of these techniques in realistic systems. Static analysis based on analytical models mostly use three general types of analysis methods including Standard Event Models (SEM), queuing-based theories and the variations such as Real-Time Calculus (RTC), automata-based models such as timed automata. Event-based models works based on periodic task models and classical scheduling algorithms. Using RTC-based methods provides more efficient approaches but still cannot describe dependencies between states. Automata-based methods model state-dependencies and provides more accurate methods [12-14].
3) Simulation-based analysis. This analysis approach works based on a simpler model of the system and a sample input trace. It reduces the evaluation time of the system but does not provide an accurate worst-case estimation. Monte Carlo is a widely used simulation-based analysis approach [15] in industry and also academia.
4) Model checking can be another alternative to verify system properties including NFPs like timing properties. However, this approach can suffer from state space explosion in case of large scale and complex real-time systems.

This work-in-progress paper proposes an adaptive runtime performance assurance in terms of timing properties. The proposed approach is a runtime technique preserving execution time of the real-time program based on an RL-based method to assure the satisfaction of timing properties.

V. Conclusion

Adaptive performance assurance through runtime monitoring of performance parameters including timing properties could be used as an extension to the core features of the supervision control program in real-time embedded systems. It will incorporate an adaptive runtime performance assurance as an extra feature in the underlying platform. The proposed learning-based timing assurance can work adaptively to uncertain temporal behaviors of the function blocks inside the real-time programs to meet the performance requirement. Furthermore, the learned policy of the runtime assurance technique can be used as feedback to re-design and correct the related details of the original properties during the Model-Driven development. Our next steps include deployment and efficiency evaluation of the proposed approach on real-time embedded systems in industry, considering the structure complexity of real-time programs and other effective/tunable parameters.

VI. Acknowledgements

This research has been funded partially by Vinnova through the ITEA TESTOMAT project and by KKS through the TOCSYC project.

References

[1] Standard Glossary of Software Engineering Terminology (ANSI), The Institute of Electrical and Electronics Engineers Inc. 1991.

[2] M. Saadatmand, M. Sjodin, and N.U. Mustafa. Monitoring capabilities of schedulers in model-driven development of real-time systems. In 17th IEEE Conference on Emerging Technologies Factory Automation (ETFA), pp. 1–10. IEEE, 2012.

[3] N. Asadi, M. Saadatmand, and M. Sjödin. Run-Time Monitoring of Timing Constraints: A Survey of Methods and Tools. In the Eighth International Conference on Software Engineering Advances, 2013.

[4] J. Huselius and J. Andersson. Model Synthesis for Real-Time Systems. In Proceedings of the Ninth European Conference on Software Maintenance and Reengineering, CSMR '05, pp. 52–60, 2005

[5] H. Thane. Design for Deterministic Monitoring of Distributed Real-Time Systems. Technical Report ISSN 1404-3041 ISRN MDHMRTC-23/2000-1-SE, Målardalen University, May 2000.

[6] M. Saadatmand, M. Sjödin. Testing of Timing Properties in Real-Time Systems: Verifying Clock Constraints. In The 20th Asia-Pacific Software Engineering Conference (APSEC), vol. 2, pp. 152-158. IEEE, 2013

[7] O. Ibidunmoye, M. H. Moghadam, E. B. Lakew, E. Elmroth. Adaptive Service Performance Control using Cooperative Fuzzy Reinforcement Learning in Virtualized Environments. In 10th IEEE/ACM International Conference on Utility and Cloud Computing, Austin, Texas, USA, 2017.

[8] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, G. Estrada. Fuzzy Self-learning Controllers for Elasticity Management in Dynamic Cloud Architectures. In 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA), pp. 70–79. IEEE, 2016.

[9] M. L. Puterman. 1994 Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc, USA

[10] R. S. Sutton, A. G Barto. 1998. Reinforcement learning: An Introduction. Vol. 1. MIT press Cambridge.

[11] International Standard IEC 61131-3, Programmable controllers-part 3 Programming languages, 2013.

[12] K. Richter and R. Ernst. Event Model Interfaces for Heterogeneous System Analysis, Design Automation and Test in Europe Conference, p. 506. IEEE Computer Society, 2002

[13] S. Chakraborty, S. Künzli, L. Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs, IEEE Design Automation & Test in Europe, vol.3, p.10190. 2003.

[14] M. Hendriks & M. Verhoef . Timed automata based analysis of embedded system architectures. In 20th International Parallel and Distributed Processing Symposium, IEEE, 2006.

[15] C. Z Mooney, *Monte carlo simulation*. Vol. 116. Sage Publications, 1997.