Mälardalen University Press Licentiate Theses
No. 289

MACHINE LEARNING-ASSISTED PERFORMANCE ASSURANCE

Mahshid Helali Moghadam

2020

School of Innovation, Design and Engineering

ISBN 978-91-7485-463-3
ISSN 1651-9256
Printed by E-Print, Stockholm, Sweden

Sammanfattning

Med det ökade beroendet av mjukvarussystem i våra liv ökar också vikten av att säkerställa systemens prestanda, eftersom detta har stor påverkan på vilken marknadsframgång produkterna kommer att ha. Olika aktiviteter som att testa prestandan, att säkerställa att den bevaras och att förbättra den bidrar alla till att säkerställa att prestandakraven uppfylls. Nuvarande metoder för att hantera utmaningar relaterade till att testa, bevara och förbättra prestanda baseras huvudsakligen på tekniker som bygger på prestandamodeller eller använder systemmodeller eller källkod. Även om modellering ger en djup inblick i systemets beteende, är det utmanande att konstruera en passande detaljerad modell som kan användas för att undersöka prestanda. En utmaning är också att artefakter som modeller och källkod inte alltid finns tillgängliga. Sammantaget motiverar detta att vi undersöker om maskininlärningstekniker som inte bygger på modeller, som till exempel modellfri förstärkningsinlärning (Reinforcement Learning, RL), för att säkerställa prestanda hos mjukvarusystem.

Avhandlingen undersöker hur RL kan tillämpas på denna typ av problem. Fördelen är att en optimal policy for att möta de avsedda målen i en prestandabevarande process kan tas fram av det agerande systemet (t.ex. testningssystemet) med maskininlärning, vilket gör att målen kan nås utan avancerade prestandamodeller. Dessutom kan den inlärda policyn senare återanvändas i liknande situationer, vilket leder till effektivitetsförbättring genom att spara beräkningstid samtidigt som man slipper beroendet av modeller och källkod.

Forskningsmålet i denna avhandling är att utveckla anpassningsbara och effektiva tekniker för att säkerställa prestandan i system och produkter utan tillgång till modeller och källkod. Vi föreslår tre anpassningsbara modellfria inlärningsbaserade metoder för att hantera de utmaningarna vi identifierat; effektiv generering av testfall för prestanda, bibehållande av prestanda (respon-

stid) och förbättring av prestanda när det gäller minskning av tiden det tar att slutföra en uppgift. Vi demonstrerar effektiviteten och anpassningsbarheten för våra tillvägagångssätt med experimentella utvärderingar. Dessa är utförda med forskningsprototypverktyg som består av de simuleringsmiljöer som vi utvecklat eller skräddarsytt för problem inom olika applikationsområden.

Abstract

With the growing involvement of software systems in our life, assurance of performance, as an important quality characteristic, rises to prominence for the success of software products. Performance testing, preservation, and improvement all contribute to the realization of performance assurance. Common approaches to tackle challenges in testing, preservation, and improvement of performance mainly involve techniques relying on performance models or using system models or source code. Although modeling provides a deep insight into the system behavior, drawing a well-detailed model is challenging. On the other hand, those artifacts such as models and source code might not be available all the time. These issues are the motivations for using model-free machine learning techniques such as model-free reinforcement learning to address the related challenges in performance assurance.

Reinforcement learning implies that if the optimal policy (way) for achieving the intended objective in a performance assurance process could instead be learned by the acting system (e.g., the tester system), then the intended objective could be accomplished without advanced performance models. Furthermore, the learned policy could later be reused in similar situations, which leads to efficiency improvement by saving computation time while reducing the dependency on the models and source code.

In this thesis, our research goal is to develop adaptive and efficient performance assurance techniques meeting the intended objectives without access to models and source code. We propose three model-free learning-based approaches to tackle the challenges; efficient generation of performance test cases, runtime performance (response time) preservation, and performance improvement in terms of makespan (completion time) reduction. We demonstrate the efficiency and adaptivity of our approaches based on experimental evaluations conducted on the research prototype tools, i.e. simulation environments that we developed or tailored for our problems, in different application areas.

To my family

Acknowledgments

This ongoing journey, as a pleasant part of my life, has been full of unique and memorable moments, and many people had supporting roles to play in different stages of the journey. First, my sincere thanks go to the great team of my supervisors. I would like to thank very much my main supervisor Prof. Markus Bohlin for his continuous support and encouragement. I am deeply grateful to Dr. Mehrdad Saadatmand for his very thoughtful technical guidance, caring, the excellence of patience, and his acts of kindness as my supervisor and manager. I am also very thankful to Dr. Markus Borg for his continuous energizing support even from a distance, helpful advice and all the great lessons that I learned from him. Last but not least, I would like to thank very much Prof. Björn Lisper for his very helpful technical comments and advice.

I have been fortunate to be a part of such an inspiring leading research organization as RISE Research Institute of Sweden and work with scientists and knowledgeable colleagues. Hereby, I would like to thank the supportive unit managers at RISE Västerås, Dr. Stig Larsson, Larisa Rizvanovic (my current manager), and Dr. Petra Edoff, and all my great colleagues. I have had the pleasure of being a part of software testing research group at RISE Västerås, which is led by Dr. Mehrdad Saadatmand, and working on interesting research projects with competent and caring colleagues, in particular Muhammad Abbass who has been always a supportive colleague/friend.

My study at Mälardalen university has provided me with the opportunity of meeting new friends and working with great people. I would also like to thank all of them. Thanks also to ITS ESS-H industrial school for all their support.

Above all, I thank God for helping me in my entire life, then I would like to express my deep gratitude to my parents, my husband, my brother, and my close friends for all their supportive presence, understanding, and being patient with me. Without their support, I would not have reached here.

Mahshid Helali Moghadam, Västerås, April 2020

List of publications

Papers included in the thesis[1]

Paper A *Machine Learning to Guide Performance Testing: An Autonomous Test Framework*, Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, Björn Lisper. The 12th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). Xian, China, April 2019.

Paper B *An Autonomous Performance Testing Framework Using Self-Adaptive Fuzzy Reinforcement Learning*, Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, Björn Lisper. Submitted to a Special Issue on Testing of Software and Systems, Software Quality Journal, February 2020.

Paper C *Intelligent Load Testing: Self-Adaptive Reinforcement Learning-Driven Load Runner*, Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Golrokh Hamidi, Markus Bohlin, Björn Lisper. Submitted to the 31st International Symposium on Software Reliability Engineering (ISSRE 2020), Coimbra, Portugal, March 2020.

Paper D *Adaptive Runtime Response Time Control in PLC-Based Real-Time Systems Using Reinforcement Learning*, Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, Björn Lisper. The 13th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2018), Gothenburg, Sweden, May 2018.

[1] The included articles have been reformatted to comply with the thesis layout.

Paper E *Makespan Reduction for Dynamic Workloads in Cluster-Based Data Grids Using Reinforcement Learning Based Scheduling*, Mahshid Helali Moghadam, Seyed Morteza Babamir. Journal of Computational Science, 24, 402-412, January 2018.

Additional papers, not included in the thesis

1. *Machine Learning-Assisted Performance Testing*, Mahshid Helali Moghadam. The 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019), Tallinn, Estonia, August 2019. **Winner of Silver Prize at ACM Student Research Competition (SRC)**

2. *Poster: Performance Testing Driven by Reinforcement Learning*, Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, Björn Lisper. The 13th IEEE International Conference on Software Testing, Verification and Validation, Porto, Portugal, March 2020.

3. *Verification-Driven Iterative Development of Cyber-Physical System*, Marjan Sirjani, Luciana Provenzano, Sara Abbaspour Asadollah, Mahshid Helali Moghadam, Mehrdad Saadatmand. Submitted to Journal of Internet Services and Applications, October 2019 (Under Revision).

4. *From Requirements to Verifiable Executable Models Using Rebeca*, Marjan Sirjani, Luciana Provenzano, Sara Abbaspour Asadollah, Mahshid Helali Moghadam. International Workshop on Automated and verifiable Software sYstem DEvelopment (ASYDE 2019), SEFM 2019 Workshop, Oslo, Norway, July 2019.

5. *Learning-Based Response Time Analysis in Real-Time Embedded Systems: A Simulation-Based Approach*, Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, Björn Lisper. The 1st IEEE/ACM International Workshop on Software Qualities and Their Dependencies (SQUADE 2018), ICSE 2018 Workshop, Gothenburg, Sweden, May 2018.

6. *Learning-Based Self-Adaptive Assurance of Timing Properties in a Real-Time Embedded System*, Mahshid Helali Moghadam, Mehrdad Saadat-

mand, Markus Borg, Markus Bohlin, Björn Lisper. The 11th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2018), Västerås, Sweden, April 2018.

7. *Adaptive Service Performance Control Using Cooperative Fuzzy Reinforcement Learning in Virtualized Environments*, Olumuyiwa Ibidunmoye, Mahshid Helali Moghadam, Ewnetu Bayuh Lakew, Erik Elmroth. The 10th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2017), Austin, Texas, USA, December 2017.

8. *A Multi-Objective Optimization Model for Data-Intensive Workflow Scheduling in Data Grids*, Mahshid Helali Moghadam, Seyyed Morteza Babamir, Meghdad Mirabi. The 41st IEEE Conference on Local Computer Networks (LCN Workshops 2016), UAE, Dubai, November 2016.

Contents

I

Thesis

Chapter 1

Introduction

With the growing dependency of different facets of our life on software, quality assurance of software with respect to both functional and non-functional aspects of behavior assumes more importance. In addition to functional correctness and completeness, one of the quality aspects which plays a significant role in success of software products is performance. In a general phrasing, it is often referred to as how well a software system (service) accomplishes the expected functionalities. Enterprise applications (EAs) [1] with internet-based user interfaces (UIs) such as e-commerce websites, banking, retailing and airline reservation systems, are examples whose success is subject to performance assurance. EAs are often the core parts of corporate business organizations and their performance mainly influences execution of business functions [2]. Internet-based EAs may receive varying number of requests from customers, and meanwhile they are required to be resilient enough upon varying execution conditions [3].

The ISO/IEC 25010 standard [4] proposes a general quality model specifying the quality characteristics of software. The quality of a software product indicates to which degree the software meets the quality requirements (needs) of the stakeholders. Performance as one of the quality characteristics in ISO/IEC 25010 quality has been also called "efficiency" in various classification schemes of quality characteristics [4, 5, 6]. Performance requirements mainly present time and resource bound constraints on the behavior of software. Those constraints are often expressed using performance metrics

such as response time, throughput, and resource utilization.

Realization of performance assurance can be viewed from different perspectives. Performance testing (evaluation), preservation and improvement all contribute to meeting performance assurance. Each aspect is associated to fulfilling some primary objectives. For example, performance testing is generally intended to meet the objectives, I. measuring performance metrics, II. detecting specific functional problems emerging under certain execution conditions such as heavy workload, III. detecting violations of non-functional requirements [7]. The execution conditions involve characteristics of the execution environment such as resource availability and characteristics of the workload under which the system operates.

Performance modeling techniques are common approaches which are widely used to meet the associated objectives in different aspects of performance assurance. Performance modeling involves building a model of system to express and measure the target performance metrics. Various modeling notations such as queueing networks, Markov processes, and petri nets [8, 9, 10] together with different analytic techniques are used for performance modeling [11, 12, 13]. Although models provide helpful insight into the performance behavior of the system, there are still many details of implementation and execution environment that might be ignored in the modeling nonetheless [14]. Moreover, building a precise model expressing the performance behavior under different and changing conditions might be difficult and costly.

In addition to performance models, many of other common approaches also rely on some artifacts such as source code or different types of system models. For example, common performance testing approaches such as techniques based on source code analysis [15], system model analysis [16, 17, 18], use case-based [19, 20], and behavior-driven [21, 22, 23, 24] design approaches mostly rely on source code or system models. Nonetheless, those artifacts might not be always available or accessible. Currently, the use of various machine learning techniques to address the challenges in different aspects of performance assurance has been frequently considered. Reinforcement Learning (RL) as a major part of machine learning is widely used for addressing decision making problems. With respect to the existing issues of common solution techniques in performance assurance, RL techniques in particular model-free RLs could play an interesting role in addressing the related challenges of performance assurance. RL algorithms are specific machine learning algorithms

in which the learning is based on the experience of interaction with system and observing the system's behavior. Model-free RLs are a subset of RL algorithms which can learn the optimal way to solve a problem (i.e., to accomplish an objective) from the interaction with the system without need to access or build a model of the system.

In Section 1.1 and 1.2 we discuss the research challenges and motivations for using RL-assisted techniques with regard to the existing issues of common techniques in performance assurance.

1.1 Research Challenges

Performance testing involves executing software under various execution conditions to measure the performance metrics and identify behavioral anomalies such as functional issues or violations of performance requirements. Verifying robustness in terms of finding performance breaking point is one of the primary purposes of performance testing. A performance breaking point often refers to a status of software at which the system becomes unresponsive or certain performance requirements (e.g., in terms of response time and error rate) get violated. In Performance testing, generating performance test cases to find the performance breaking point is a challenge for complex systems. Furthermore, similarly preserving the performance of a system or optimizing it with respect to changeable execution conditions is also challenging.

Performance model-driven techniques for addressing challenges in the scope of testing, preserving and optimizing performance, mainly suffer due to the costly process of building detailed performance models, in particular for complex systems. Relying on source code and system models in other approaches and also some issues like the need for automated and efficient generation of performance test cases (See Chapter 3) are some of the major raised issues in connection to the existing common techniques.

Regarding the aforementioned issues, we propose that reinforcement learning techniques could help tackle the challenges without relying on source code or model artifacts. In this thesis, we discuss various aspects of performance assurance, mainly performance testing, and present how model-free reinforcement learning can guide them towards finding the optimal way of accomplishing the objectives, and meanwhile alleviate the dependency on models and source code. Moreover, we show how the capability of reusing knowledge can

improve the efficiency of the activity in terms of reduced required effort, i.e., time and cost.

1.2 Motivation

Performance testing (evaluation) as one of the main steps towards performance assurance, is important for performance-critical software systems in various domains. Performance anomalies and violations of performance requirements are generally consequences of performance bottlenecks [25, 26]. A performance bottleneck is a system or resource component limiting the performance of the system and making the system fail to act as well as required [27].

The occurrence of some limitations associated with the component such as saturation and contention makes a component act as a bottleneck. A system or resource component saturation happens upon full utilization of its capacity or exceeding a usage threshold [27]. The primary causes of performance bottleneck emergence can be categorized into three groups, application-based, platform-based, and workload-based ones. Application-based causes are the issues such as defects in the source code or system architecture faults, while the issues related to hardware resources, operating system, and execution environment could be described as platform-based causes. Workload-based causes also represent the issues such as deviations from the expected workload intensity.

Therefore, for example to address the challenge of performance testing with the purpose of finding performance breaking point, we need to find how to provide critical execution conditions which make the performance bottlenecks emerge. The focus of performance testing in our research is to assess the robustness of system and find the performance breaking point. We want to achieve this based on generating platform-based and workload-based test conditions in this research thesis.

Regarding this objective, it is required to keep in mind that the effects of internal causes, i.e., application/architecture-based ones, are also important, and they could vary due to continuous changes and updates of the software, i.e., Continuous Integration/Continuous Delivery (CI/CD). They might act differently on different platforms (execution environments) and under different workload conditions. Therefore, in many cases it is hard to build a precise per-

formance model expressing the effects of all factors at play due to the complexity of the software under analysis/test (SUT) and the internal affecting factors.

It can be inferred that the same conditions and challenges are valid for other tasks like performance preservation or performance improvement. This is a major barrier motivating the use of model-free learning-based approaches like model-free RL, in which the optimal policy for accomplishing the objective could be learned indirectly through interaction with the environment (software) and reused in further similar situations. In our problem statement, we consider the software (i.e., with its involved factors affecting performance) as the environment. Then, the learning system, which is an smart agent and does testing, control or improvement, explores the behavior of the environment and learns the optimal policy to achieve the intended target, without access to source code or a model of the environment. It stores the learned policy and is able to later reuse the learned policy in similar situations. This is an interesting feature of the proposed learning approach which is supposed to lead to productivity benefits (reduction of computation time) in the problem.

1.3 Research Process

In order to conduct research in a right way, using a proper research methodology is of great importance. G. Dodig-Crnkovic [28] describes a scientific framework which is widely used as a logical scheme by researchers and scientists to address research questions in general sciences. She discusses scientific differentiating aspects of computer science from other sciences and describes how the general scientific methodology can be customized for computer science fields. H. J. Holz et al. [29] also provide an overview of different computing research methods and a general framework for organizing the process of computing research. Their framework involves four steps describing and handling the cycle of research collectively. The four steps, identifying research problem (challenge), formulating research goal/question, proposing a solution, and evaluating the proposed solution (i.e., followed by an industrial validation step in some cases) are the main ones inspired from their framework.

Regarding the industrial validation step, it is worth remarking that there are always challenges for software engineering research in reaching its potential due to the lack of enough research grounded on realistic application contexts. The solutions which do not match the real needs and could not scale are of those

challenges [30]. Therefore, we customize the general framework inspired from [29], as shown in Figure 1.1.

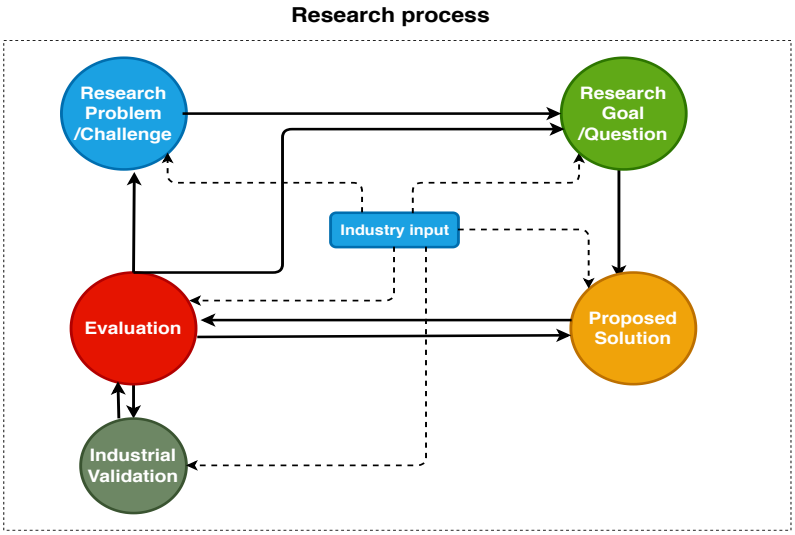


Figure 1.1: Research process

We try to consider the feedback from the industry's point of view in each step. Thus, the realization of each step in our research is summarized as follows:

Research Problem. We identify the research problems based on reviewing both state of the art and practice. We review the literature and find the sources, in particular through the search methods of systematic literature review (SLR) such as snowballing.

Research Goal/Question. The research goals or questions are formulated based on the identified research problems (challenges).

Proposed Solution. After identifying the research problems and formulating research goals, we consolidate our ideas for meeting the research goals in terms of initial solutions. Then, in further steps based on our studies and received feedback from industry's point of view, we improve the initial solutions and develop the improved ones (i.e., as research prototypes) which can be used for the evaluation step.

Evaluation. We choose experimentation as an empirical research method, based on the guidelines provided by Robson and McCartan [31] for the evaluation. In the evaluation step, we evaluate the efficacy and efficiency of our solutions through conducting a set of controlled experiments in accordance with the existing guidelines [32]. Depending on the evaluation results, the research problems, goals, and the proposed solutions could be refined. This process can be conducted iteratively until reaching the desired results.

Moreover, an industrial validation of the developed research prototype could be conducted based on the industry's view on the solution.

1.4 Research Goals

The main challenge that we address in this thesis is accomplishing performance assurance objectives from the perspectives of performance testing, preservation and improvement, particularly in the cases where performance models are not available or building a model is too costly. We investigate the use of model-free reinforcement learning techniques which learn the optimal way of meeting the objectives without access to model or source code and provide the opportunity to transfer the gained knowledge between similar situations. Therefore, the main research goal driving this research is as follows:

Overall Research Goal. To introduce and develop adaptive learning-based **performance assurance** techniques that are able to learn the optimal way (policy) to meet the intended objectives and re-use the knowledge properly in potential situations, without access to underlying models or source code.

The general theme of the overall research goal is solution-focused [33], i.e., it focuses on creating better (more effective and efficient) solutions. The overall research goal is divided into three subgoals. They are as follows:

- **subgoal 1:** To formulate and develop a self-adaptive model-free performance **testing** framework that is able to learn the efficient generation of the performance test cases to meet the intended testing objectives in different testing situations.
- **subgoal 2:** To formulate and develop an adaptive performance **preservation** technique able to learn how to keep the target performance requirement satisfied in changing conditions.

- **subgoal 3:** To design and develop an adaptive model-free learning-based technique for performance **improvement**.

Figure 1.2 shows how each step of the research process has been realized in the thesis.

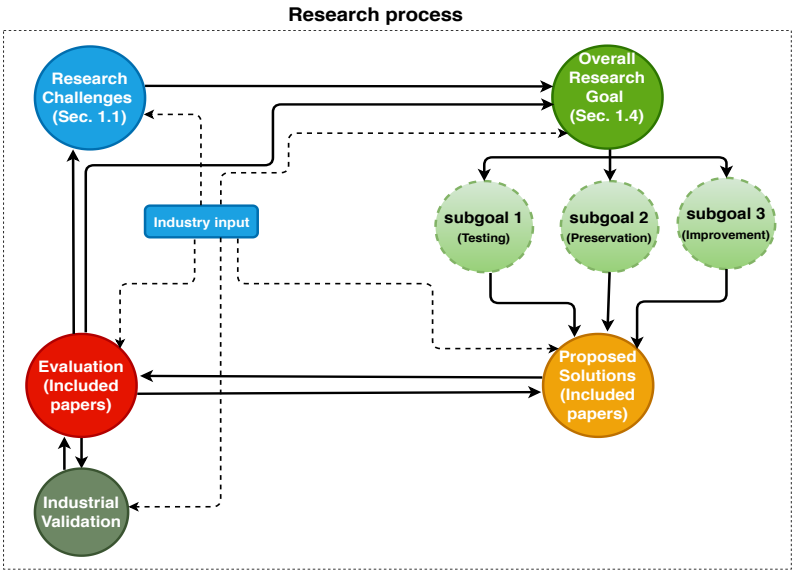


Figure 1.2: Realization of research steps in the thesis

During this research, we have chosen different application domains to accomplish the research subgoals. In the remainder of the thesis, we present our contributions in relation to the research subgoals mentioned above. Note that we have mostly used simulation methods to carry out the experimental evaluations in our contributions. Our simulation-based experimental evaluation allowed us to show the efficacy and applicability of RL-assisted techniques in different aspects of performance assurance for meeting the related objectives. There are also specific heuristics and customized techniques which could facilitate the use of RL techniques in realistic scaled-up environments with more complexity.

1.5 Thesis Outline

This thesis is divided into two parts. The first part is a summary of the thesis and is organized in four chapters, which are as follows: Chapter 1 gives an overview of the preliminaries, research challenges, research goals, motivations, and the research process which directed our research. In Chapter 2, we describe the contributions of the thesis to realization of the research goals. Chapter 3 presents an overview of the related work and background concepts, in particular RL algorithms. Finally, in Chapter 4, we conclude the first part of the thesis with a discussion on our results as well as possible directions for the future work. The second part of the thesis is given as a collection of the included publications which present the technical contributions of the thesis in detail.

Chapter 2

Research Contribution

In this section we summarize our contributions in the thesis to achieve the research goals mentioned in Section 1.4.

Contribution towards subgoal 1 (C1): The contributions towards achieving subgoal 1 consist of three parts, C1.1, C1.2, and C1.3, which are described as follows:

C1.1. Formulation of the performance testing problem in terms of an RL problem. First, we investigate the possibility of applying RL algorithms to a performance testing problem with the purpose of finding performance breaking points. We propose an initial architecture of a learning-based performance testing agent and present a general overview of the main parts of the architecture and how each step of the learning is formulated (paper A) [34]. Q-learning [35], which is a well-known reinforcement learning, is used as the core learning procedure in the proposed approach. The proposed RL-based smart agent basically learns the optimal policy of accomplishing the intended objective (i.e., reaching the intended performance breaking point) through episodes of interaction with the environment, i.e., SUT and execution platform. This interaction generally involves sensing the state of the environment, taking an action which affects the environment towards the intended objective and receiving a reward signal which shows the effectiveness of the applied action. The primary idea of a smart tester agent is formulated as follows:

How the smart agent works. We use Q-learning as the core learning algorithm in the smart agent. The proposed smart tester agent assumes two phases of learning:

- Initial learning during which the agent learns an optimal policy for the first time.
- Transfer learning during which the agent replays the learned policy in similar cases while keeping the learning running in the long term.

The initial proposed architecture (paper A) uses Q-learning together with the idea of using multiple experience (knowledge) bases. It stores the learned optimal policy of achieving the objective (i.e., finding performance breaking point) for different types of SUT, i.e., CPU-intensive, memory-intensive and disk-intensive programs in separate knowledge bases. This paper uses experience adaptation based on using multiple experience bases during the learning.

Subsequently, we extend the primary idea of RL-assisted performance testing and develop a self-adaptive model-free reinforcement learning-driven performance testing framework mainly involving two parts which are as follows:

- SaFReL: self-adaptive fuzzy reinforcement learning performance testing through platform-based test cases (paper B)
- RELOAD: adaptive reinforcement learning-driven load testing (paper C)

C1.2. SaFReL. We extend and improve the initial concept proposed in paper A and develop a self-adaptive fuzzy reinforcement learning testing agent, SaFReL. It generates the platform-based performance test cases resulting in the intended performance breaking point, for different software programs without access to source code or system models (paper B). The proposed smart tester agent assumes two learning phases, initial and transfer learning, as described above. We augment the learning by adopting fuzzy logic to model the state space of the environment (SUT and execution platform) and fuzzy classification for state detection. It helps tackle the issue of uncertainty in defining discrete classes and improve the accuracy of the learning. We also propose an adaptive action selection strategy adjusting the parameters related to the action selection based on the detected similarity between the performance sensitivity of SUTs. It is intended to make the learning adapt to different testing cases.

We conduct a two-fold experimental evaluation, i.e., performance (efficiency and adaptivity) and sensitivity analysis of the testing agent. The evaluation is carried out based on simulating performance behavior of various SUTs. We implement a performance prediction module along with our smart tester agent to conduct the experimentation. According to our experimental evaluation, SaFReL meets the objective, i.e., reaches the intended performance breaking point, more efficiently in comparison to a typical performance testing technique which mainly generates the performance test cases based on changing the performance test conditions, by certain steps in an exploratory way. SaFReL leads to a reduced cost in terms of computation time for performance test case generation by reusing the learned policy upon the SUTs with similar performance sensitivity. Moreover, it adapts its operational strategy to various SUTs with different performance sensitivity effectively while preserving the efficiency.

C1.3. RELOAD. The second part of the smart framework is an intelligent RL-assisted load testing agent which learns the optimal policy to generate a workload resulting in meeting an intended error rate threshold (paper C). It basically uses Q-learning with an adaptive action selection strategy to improve the learning performance. The intelligent load testing agent, RELOAD, uses a load generator/runner tool (i.e., Apache JMeter) to execute the recommended workload on the SUT. We use a typical e-commerce store running on a WordPress hosting server as SUT in the experimental evaluation. According to the results, RELOAD generates a more efficient workload (i.e., smaller workload in terms of number of users) to meet the intended objective compared to a typical load testing technique which mainly involves applying a basic workload and increasing the load of the involved transactions by a certain increase step. The generation of an efficient test workload without dependency on source code, system and user behavior models, and also reduction of the required effort for workload generation in further situations (e.g., regression load testing) are the main strengths of the proposed RL-driven load testing

Contribution towards subgoal 2 (C2): We formulate an adaptive RL-assisted performance preservation technique, i.e., an adaptive runtime response time control for PLC-based programs, and conduct an experimental evaluation using simulating the performance behavior of the programs (paper

D) [36]. The proposed approach formulates a model-free RL-assisted response time control using Q-learning to provide adaptive preservation of response time according to the timing requirement. We evaluate the efficacy of the approach through multiple experiments. Our approach mostly keeps the programs adhering to the response time requirements despite various execution conditions during the run time, i.e., the occurrence of time-related events resulting in time deviations.

Contribution towards subgoal 3 (C3): Our contribution is based on performance improvement in terms of makespan reduction for data-intensive task execution in a data grid. We propose a two-step model-free learning-based task placement (i.e., resource allocation) for executing data-intensive tasks (paper E) [37]. We conduct the evaluation experiments based on integrating our proposed technique into an open source simulation environment which we used in the experiments. We evaluate the efficacy of our approach in terms of resulted makespan (completion time) for submitted tasks. We investigate the achieved performance improvement in our learning technique under different types of workloads. Our proposed learning-based algorithm results in performance improvement in comparison to four common algorithms which mainly use parameters of the environment and work based on model analysis of the environment.

2.1 Overview of the Included Papers

The main contributions of the thesis are organized and presented as a set of papers which have been included in the thesis. Other papers that have been just listed at the beginning of the thesis and are not included, also strengthen the contributions of the thesis. A summary of the included papers is as follows:

- Paper A: Machine Learning to Guide Performance Testing: An Autonomous Test Framework

  Abstract: Satisfying performance requirements is of great importance for performance-critical software systems. Performance analysis to provide an estimation of performance indices and ascertain whether

the requirements are met is essential for achieving this target. Model-based analysis as a common approach might provide useful information but inferring a precise performance model is challenging, especially for complex systems. Performance testing is considered as a dynamic approach for doing performance analysis. In this work-in-progress paper, we propose a self-adaptive learning-based test framework which learns how to apply stress testing as one aspect of performance testing on various software systems to find the performance breaking point. It learns the optimal policy of generating stress test cases for different types of software systems, then replays the learned policy to generate the test cases with less required effort. Our study indicates that the proposed learning-based framework could be applied to different types of software systems and guides towards autonomous performance testing.
Contribution: I have been the initiator and main author of the paper.

- Paper B: An Autonomous Performance Testing Framework Using Self-Adaptive Fuzzy Reinforcement Learning

  Abstract: Test automation brings the potential to reduce costs and human effort, but several aspects of software testing remain challenging to automate. One such example is automated performance testing to find performance breaking points. Current approaches to tackle automated generation of performance test cases mainly involve using source code or system model analysis or use-case based techniques. However, source code and system models might not always be available at testing time. On the other hand, if the optimal performance testing policy for the intended objective in a testing process instead could be learned by the testing system, then test automation without advanced performance models could be possible. Furthermore, the learned policy could later be reused for similar software systems under test, thus leading to higher test efficiency. We propose SaFReL, a self-adaptive fuzzy reinforcement learning-based performance testing framework. SaFReL learns the optimal policy to generate performance test cases through an initial learning phase, then reuses it during a transfer learning phase, while keeping the learning running and updating the policy in the

long term. Through multiple experiments on a simulated environment, we demonstrate that our approach generates the target performance test cases for different programs more efficiently than a typical testing process, and performs adaptively without access to source code and performance models.
Contribution: I have been the initiator and main author of the paper.

- Paper C: Intelligent Load Testing: Self-Adaptive Reinforcement Learning-Driven Load Runner

  Abstract: Load testing with the aim of generating an effective workload to identify performance issues is a time-consuming and complex challenge, particularly for evolving software systems. Current automated approaches mainly rely on analyzing system models and source code, or modeling of the real system usage. However, that information might not be available all the time or obtaining it might require considerable effort. On the other hand, if the optimal policy for generating the proper test workload resulting in meeting the objectives of the testing can be learned by the testing system, testing would be possible without access to system models or source code. We propose a self-adaptive reinforcement learning-driven load testing agent that learns the optimal policy for test workload generation. The agent can reuse the learned policy in subsequent testing activities such as meeting different testing targets. It generates an efficient test workload resulting in meeting the objective of the testing adaptively without access to system models or source code. Our experimental evaluation shows that the proposed self-adaptive intelligent load testing can reach the testing objective with lower cost in terms of the workload size, i.e., the number of generated users, compared to a typical load testing process, and results in productivity benefits in terms of higher efficiency.
  Contribution: I have been the initiator and main author of the paper. A part of the experiments has been implemented by Golrokh Hamidi.

- Paper D: Adaptive Runtime Response Time Control in PLC-Based Real-Time Systems Using Reinforcement Learning

  Abstract: Timing requirements such as constraints on response

time are key characteristics of real-time systems and violations of these requirements might cause a total failure, particularly in hard real-time systems. Runtime monitoring of the system properties is of great importance to check the system status and mitigate such failures. Thus, a runtime control to preserve the system properties could improve the robustness of the system with respect to timing violations. Common control approaches may require a precise analytical model of the system which is difficult to be provided at design time. Reinforcement learning is a promising technique to provide adaptive model-free control when the environment is stochastic, and the control problem could be formulated as a Markov Decision Process. In this paper, we propose an adaptive runtime control using reinforcement learning for real-time programs based on Programmable Logic Controllers (PLCs), to meet the response time requirements. We demonstrate through multiple experiments that our approach could control the response time efficiently to satisfy the timing requirements.
Contribution: I have been the initiator and main author of the paper.

- Paper E: Makespan Reduction for Dynamic Workloads in Cluster-based Data Grids Using Reinforcement Learning Based Scheduling

  Abstract: Scheduling is one of the important problems within the scope of control and management in grid and cloud-based systems. Data grid still as a primary solution to process data-intensive tasks, deals with managing large amounts of distributed data in multiple nodes. In this paper, a two-phase learning-based scheduling is proposed for data-intensive tasks scheduling in cluster-based data grids. In the proposed approach, a hierarchical multi agent system, consisting of one global broker agent and several local agents, is applied to scheduling procedure in the cluster-based data grids. At the first step of the proposed approach, the global broker agent selects the cluster with the minimum data cost based on the data communication cost measure, then an adaptive policy based on Q-learning is used by the local agent of the selected cluster to schedule the task to the proper node of the cluster. The impacts of three action selection strategies have been investigated in the proposed approach, and the performance of different versions of

the approach regarding different action selection strategies, has been evaluated under three types of workloads with heterogeneous tasks. Experimental results show that for dynamic workloads with varying task submission patterns, the proposed learning-based scheduling gives better performance compared to four common scheduling strategies, Queue Length (Shortest Queue), Access Cost, Queue Access Cost (QAC) and HCS, which use regular combinations of primary parameters such as, data communication cost and queue length. Applying a learning-based strategy provides the scheduling with more adaptability to the changing conditions in the environment.
Contribution: I have been the initiator and main author of the paper.

The contributions of the papers to reach the subgoals are summarized in Table 2.1.

Table 2.1: Mapping of the papers to the contributions and research goals

Thesis Paper	**Contribution**	**Research Goal**
A	C1.1	G1
B	C1.2	G1
C	C1.3	G1
D	C2	G2
E	C3	G3

Chapter 3

Background and Related Work

This chapter discusses reinforcement learning, in particular model-free reinforcement learning algorithms and the role of them in finding the optimal behavior to meet an objective in learning problems. In later sections, it presents a cross-section of the related work.

3.1 Reinforcement Learning

The concept of reinforcement learning (RL) [35] can be found in various sections in different fields of science. It is intended to solve the problems of decision making. In fact, decision making is the common nature of all the problems to which RL is going to address. In general, RL is a fundamental science which is intended to find the optimal way to make decisions. Therefore, many faces of this concept can be found in many different fields of science in which same problems are solved with similar techniques under different names, such as optimal control techniques in engineering fields. Reinforcement learning in computer science belongs to the field of machine learning.

Machine learning is an analytical technique which does the same process as the learning process in the human mind. Generally, there are three main types of machine learning techniques as supervised, unsupervised and reinforcement

learning. Supervised learning mainly involves building a model and extracting useful patterns from a training data set including known input and output. The extracted model is often used for prediction purposes. The supervised learning algorithms work based on classification or regression. Classification techniques mainly build models on discrete data while regression techniques are used to produce/predict continuous output. A number of common classification algorithms are k Nearest Neighbor (KNN), Support Vector Machine (SVM), Neural Network, Naiive Bayes and Decision Trees [38, 39]. Some of the common regression techniques are Gaussian Process Regression (GPR) models, SVM regression, regression trees and generalized linear models. Unsupervised learning explores data to find hidden patters/structures. Clustering techniques are the most common algorithms in the category of unsupervised learning. A number of common clustering algorithms are K-means, K-Medois, hierarchical clustering, Fuzzy c-means and Gaussian mixture models [40, 41, 38].

Reinforcement learning (RL) is a different learning paradigm from the previously mentioned ones, which is based on interaction with the environment (system [1]) of the problem. Basically, at each step of the interaction, the agent observes (senses) the environment, takes a possible action and receives a reward signal from the environment showing the effectiveness of the applied action to accomplish the intended objective of the agent (Figure 3.1). Some of the major distinct differences between RL and other learning paradigms are as follows [35, 42]:

- There is no supervisor at play in RL, the agent just receives a reward signal.
- The agent receives the reward (feedback) with a delay.
- RL is based on sequential decision making process. The learning does not occur based on a training data set. Instead, the agent goes through the environment, decides at each step and based on optimizing the reward, it learns the optimal way of decision making.
- The agent's actions influence the system and consequently affect the subsequent received data by the agent.

[1] system and environment are used interchangeably hereafter in the thesis.

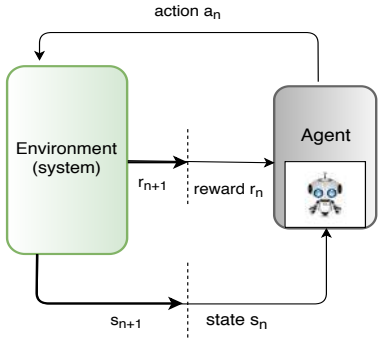


Figure 3.1: Reinforcement learning cycle between agent and environment

3.1.1 Principles

The principle concepts in RL are as follows [35, 42]:

State. In general, the agent takes actions based on its observations from the environment (system). In fact, at each step, the agent decides what actions to take based on the history. The history refers to the sequence of observations, actions and rewards that happened during the past steps. Considering the whole history is not definitely efficient, therefore, *state* as a concise summary of the history, which includes all the required information, is used to determine what should be taken next.

One related concept to the (summary) function of the history, is the concept of *Markov state*. A state S_t is Markov if and only if

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_t, S_{t-1}, ..., S_1] \tag{3.1}$$

The states within the environment are Markov by definition. If the environment is fully observable to the agent, then the states for the agent (the agent's states), which are used for making decisions on the actions, are Markov too. This representation shows the main formalism for RL which is Markov Decision Process (MDP). If the environment is partially observable to the agent, which means that the agent does not observe the whole environment, then the states for the agent are not equivalent to the environment's states. This situation needs a different formalism for RL, such as *partially observable Markov Decision Process* and certain heuristics such as keeping the whole history or

using a recurrent neural network, could be used to build the states of the agent. The environments (systems) in the cases in this thesis are assumed to be full observable to the agent.

Action and Reward. The agent selects the actions to maximize the long-term reward. The reward is a scalar feedback signal which shows how well the agent does at each step. Generally, the goal of the agent is maximizing the cumulative long-term reward.

Agent's Properties. An agent which acts based on RL might have one or some of the following learning elements:

- Policy which is a function describing the behavior of the agent, i.e., what actions the agent selects given a certain state (a map from states to actions).
- Value function which describes how good each state and/or action is. In other words, how much reward we expect upon taking a particular action in a particular state (a prediction of future reward). Figure 3.2 shows an example of the state value and policy map of an RL problem in a Maze example.
- Model which is the agent's representation (view) of the environment. It predicts what the environment will be next. The agent might have (or build) a model of the environment. Two types of models, i.e. transition-based and reward-based models, are often used to show the behavior of the environment. A transition-based model predicts the next state given a certain state and taking a particular action and a reward-based gives the next immediate reward upon taking a particular action in a particular state. Many of the Rl algorithms do not use a model of the environment (model-free RL).

3.1.2 RL Algorithms

Reinforcement learning algorithms could be categorized based on the learning elements that the agent uses. A fundamental categorization of RL algorithms is as follows [35]:

Model-free. These RL algorithms are not intended to explicitly build or learn a model of the environment to understand how the environment works.

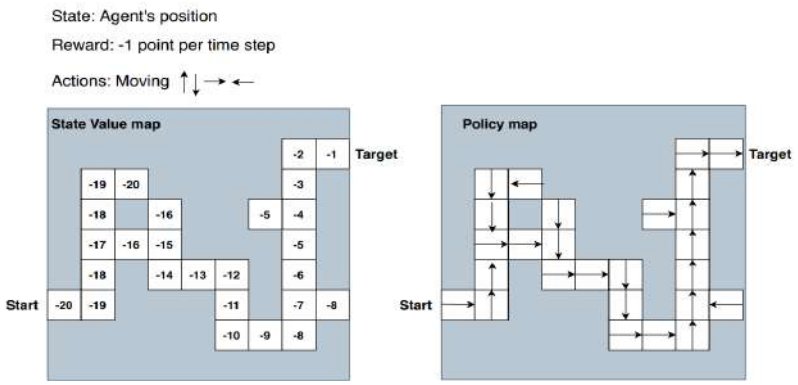


Figure 3.2: An example of the state value and policy map of an RL problem (Maze example)

The purpose of these algorithms are learning the optimal behavior, i.e., understanding how to behave to achieve as much reward as possible through multiple experiences of interaction with the environment. Monte Carlo learning and Temporal-Difference (TD) learning including Q-learning algorithms are well-known model-free RL algorithms.

Model-based. These algorithms first build a model (dynamics) of the environment, then use the model for planning and looking ahead to find out the optimal way to behave.

Other possible categorizations based on the key elements of the learning are value-based, policy-based and actor critic algorithms. A value-based algorithm just uses the value function and the policy can be implicitly read and extracted. A policy-based algorithm instead of using value-functions, stores and uses the policy explicitly. Finally, an actor critic algorithm stores and uses both the value-function and policy together.

3.1.3 Model-Free RL Algorithms

Access to a model of the environment could be an unrealistic assumption for many real complex systems. In model-free RL algorithms, there is no assumption of access to a model of the environment, i.e., no known MDP of the environment. Instead, they use the experience of interactions with the system to

estimate the value function and hence find the optimal policy to accomplish the intended objective of the agent. There are two directions of using model-free RL algorithms which are estimating the value function of a given policy and optimizing the value function (finding the optimal value function) which results in the optimal policy [35].

Monte-Carlo (MC) RL is the first major family of model-free methods, which might not be the most efficient methods but are very effective and widely used in practice. MC techniques use the complete (terminated) episodes of the interaction with the environment to estimate the value function of states given a particular policy. The value function of state s under policy π, $V_\pi(s)$, is the expected return (Equation 3.2) from state s under policy π, which is indicated as follows:

$$G_t = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{T-1} r_T \tag{3.2}$$

$$V_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] \tag{3.3}$$

where r_{t+1} is the received reward upon first transition from state s, γ is the discount factor, and G_t is the return (total discounted reward till the ending of episode). For estimating the value function of a particular state, MC uses the mean of the returns of the experienced episodes started from that particular state. However, a mean value could be calculated incrementally (See Equation 3.4 which shows the incremental calculation of μ_n representing the average value of $x_1, x_2, ..., x_n$). Therefore, the average value of returns could be similarly computed episode by episode without the need to keep the sum of the episodes. Therefore, the incremental updates in Monte-Carlo technique is as Equation 3.5.

$$\begin{aligned} \mu_n &= \frac{1}{n} \sum_{i=1}^{n} x_i \\ &= \frac{1}{n}(x_n + (n-1)\mu_{n-1}) \\ &= \mu_{n-1} + \frac{1}{n}(x_n - \mu_{n-1}) \end{aligned} \tag{3.4}$$

$$\begin{aligned} N(s) &= N(s) + 1 \\ V(s) &= V(s) + \frac{1}{N(s)}(G_s - V(s)) \end{aligned} \tag{3.5}$$

where $N(s)$ indicates the number of visits to state s, G_s represents the return resulted from state s in the current episode and $V(s)$ is the previous estimate of $V(s)$. There are also two ways to count the number of visits to state s in an episode and update the value, which are either the *First time* or *Every time* that state s is visited in the episode. Finally, in order to avoid keeping the whole statistics about the episodes, the number of state visits could be replaced with a constant step size, α, and the update equation could be written as follow:

$$V(s) = V(s) + \alpha(G_s - V(s)) \tag{3.6}$$

Temporal-Difference (TD) learning. This set of algorithms also learns from the experienced episodes of the interaction similarly. However, the distinct difference between MC and TD methods is that TD learns from the *incomplete* episodes (trajectories) by doing *bootstrapping*, i.e., using an estimate in place of actual reward, in the incremental updating. It is an online learning method since it learns the state value under policy π in an online way, i.e., in each step of the experience of interaction with the environment.

The simplest kind of TD learning, TD(0), uses an estimate of the return in one step ahead and updates the state value incrementally as follows:

$$V(s_t) = V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \tag{3.7}$$

where r_{t+1} is the received reward upon transition to s_{t+1}, then $r_{t+1} + \gamma V(s_{t+1})$ is the estimated target return, *TD target*, and subsequently $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ is the *TD error*. Some of the major benefits of TD compared to MC are as follows:

- TD can learn online after every step without need to wait until the end of episodes and is able to learn from incomplete episodes, while MC must wait for the completed episodes.
- TD can work in continuing environments, while MC is able to work only in terminating episodic environments. This feature makes TD proper to changeable non-terminating environments.

- TD works more efficiently than MC due to doing bootstrapping.

One of the main features of TD is that it converges to a solution connected to an MDP with max likelihood. It means that it uses Markov property, finds an MDP that fits the observations best and tries to solve it. Therefore, this feature makes TD able to converge even on a number of repeated sample episodes, while MC does not act in the same way. MC does not exploit Markov property, then it could be more effective in non-Markov environments.

As mentioned before, TD(0) takes one step of the reality to make an estimate of the state value, while it is possible to make TD look into more steps of the reality and then make a better estimate. TD(n) is intended to use n-step return (See Equation 3.8) for incremental updates as follows:

$$G_{s_t}^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}) \tag{3.8}$$

$$V(s_t) = V(s_t) + \alpha(G_{s_t}^{(n)} - V(s_t)) \tag{3.9}$$

TD(λ) is a way which tries to use n-step returns from all time-steps efficiently. It uses a geometrically weighted average of all n-step returns using a constant weight λ, $0 \leq \lambda \leq 1$, which is as follows:

$$G_{s_t}^{(\lambda)} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{s_t}^{(n)} \tag{3.10}$$

Then, the following equation provides *Forward-view* $TD(\lambda)$ using weighted average, $G_s^{(\lambda)}$:

$$V(s_t) = V(s_t) + \alpha(G_{s_t}^{(\lambda)} - V(s_t)) \tag{3.11}$$

However, Forward-view $TD(\lambda)$ again suffers from the same disadvantages as MC because it can be computed only from completed episodes. Thus, there is another way, *Backward-view* $TD(\lambda)$, which provides a mechanism to provide the possibility of online, every step updates from incomplete episodes to $TD(\lambda)$. It uses a simple function, $E_t(s)$, for keeping an eligibility trace for every state that has been visited (See Equation 3.16). It uses both frequency and recency heuristics to give credits to the states.

$$\begin{aligned} E_0(s) &= 0 \\ E_t(s) &= \gamma\lambda E_{t-1}(s) + 1 \end{aligned} \tag{3.12}$$

Therefore, backward-view $TD(\lambda)$ updates the state value based on one-step TD-error and eligibility trace, which is as follows:

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \\ V(s_t) &= V(s_t) + \alpha \delta_t E_t(s) \end{aligned} \tag{3.13}$$

3.1.4 Model-Free RL for Optimal Behavior

The main motivations for using model-free RL algorithms for finding the optimal policy to behave could be generally described as follows [35, 42]:
First, the MDP model of the environment might be unknown, however sampling experience is possible. Second, the MDP model might be known but it's too complex and computation-intensive to use, thus it's more efficient to use samples of the environment.

This section presents how core model-free RL algorithms (See Section 3.1.3) are used to figure out the optimal policy to reach a target without having any information on how the environment works. In Section 3.1.3, it was discussed that how the core model-free RL algorithms, Monte Carlo and TD learning, evaluated a given policy, i.e., estimated the value function of a particular policy in an unknown MDP. In this section, we discuss how the aforementioned estimation technique together with an improvement strategy could be used to optimise the value function in an unknown MDP.

In general there are two paradigms as on-policy and off-policy learning for learning the optimal behavior to reach an intended target. On-policy learning finds the optimal policy through optimising the policy from which the experience samples are, while off-policy learning learns the optimal policy from the experience samples which have been produced based on other policies, e.g., from the others' behavior. The main idea behind these learning paradigms is using a policy iteration process consisting of iterative policy evaluation and iterative policy improvement. In fact, at each step, the agent alternates between policy evaluation and improvement, as first it evaluates the policy (estimate the value function), then tries to improve the policy through a greedy approach. This process will converge on optimal value function and policy (See Figure 3.3).

However, the important issue related to using policy iteration is that for applying a greedy approach to improve the state value function, we need an

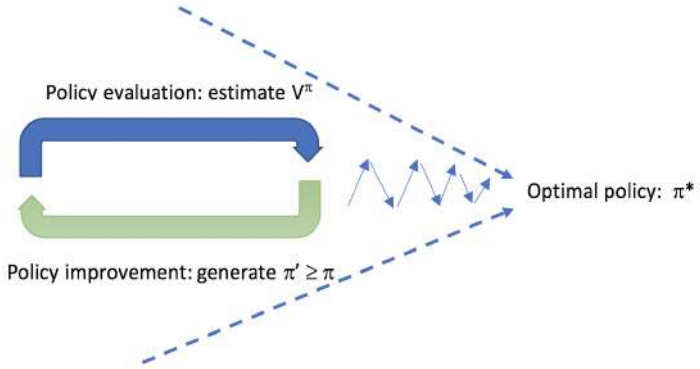


Figure 3.3: Policy iteration towards optimal policy

MDP model, while we do not have access to MDP model in model-free learning. Therefore, using action-value function, $Q(s,a)$, instead of $V(s)$ is a way to address the issue and make the greedy policy improvement possible (See Equation 3.14)

$$\pi'(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a) \tag{3.14}$$

Moreover, meanwhile in order to assure the possibility of exploration, ε-greedy, $0 < \varepsilon < 1$ is used as policy improvement approach which indicates that a greedy action, $\underset{a \in A}{\operatorname{argmax}}\, Q(s,a)$, is chosen with probability $1 - \varepsilon$, otherwise a random action is selected.

Both Monte-Carlo and TD algorithms can be used in the aforementioned policy iteration process. However due to the basic strengths of TD such as online update, the capability of working in continual environments, and learning from incomplete episodes, using TD in the policy iteration process to figure out the optimal policy is more common and efficient. Therefore, the general idea is using TD to evaluate $Q(s,a)$, using ε-greedy policy improvement and updating upon each time-step. The update is done based on *Sarsa* updating rule, which considers an estimate of policy in one step ahead and updates the Q-value of the current state in the direction of the estimate, which is as follows:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{3.15}$$

where s' is the next state and a' is the action which is taken at the next

state. Sarsa algorithm (See Algorithm 1) is the primary on-policy model-free algorithm based on the use of TD and Equation 3.15 to find the optimal policy to reach a target.

Algorithm 1 Sarsa Algorithm

Initialize Q-values, $\forall s \in \mathbb{S},\ \forall a \in \mathbb{A}$;
while *Not (end of learning)* **do**
    Initialize s;
    Choose action a based on the policy derived from Q (e.g., using ε-greedy);
    repeat
        Take action a;
        Observe s', r;
        Choose action a' based on the policy derived from Q (e.g., using ε-greedy);
        $Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$;
        $s \leftarrow s'\ a \leftarrow a'$;
    until *meeting the terminal (target) state*;
end

Moreover, Sarsa can also benefit from the idea related to TD(n), considering the n-step returns, similarly and updates $Q(s,a)$ in the direction of n-step Q-returns. Then, backward-view Sarsa(λ) like TD(λ) uses an eligibility trace for each pair of state and action, which is as follows:

$$\begin{aligned} E_0(s,a) &= 0 \\ E_t(s,a) &= \gamma\lambda E_{t-1}(s,a) + 1(s_t = s, a_t = a) \end{aligned} \tag{3.16}$$

Upon visiting each state-action pair, its eligibility increases by one and the eligibility of others decays. Moreover, Sarsa (λ) updates not only the $Q(s,a)$, which has been visited, but also all the Q-values of all other state-action pairs in proportion to TD-error and eligibility trace (See Algorithm 2).

In addition to the on-policy learning, off-policy learning is a similar type of learning featuring in the capability of learning from observing others' behavior, which is a real advantage over the on-policy learning. It is able to reuse the previous experiences directed by old policies and even learn the optimal policy while it follows an exploratory policy.

Algorithm 2 Sarsa(λ) Algorithm

Initialize Q-values, $\forall s \in \mathbb{S},\ \forall a \in \mathbb{A}$;
while *Not (end of learning)* **do**
  Initialize S , A
  Choose action A based on the policy derived from Q-values (e.g., using ε-greedy)
  repeat
    Take action A
    Observe S', r
    Choose action A' based on the derived policy (e.g., using ε-greedy)
    $\delta = r + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) = E(S, A) + 1$
    For all $s \in \mathbb{S},\ a \in \mathbb{A}$
    $\{\ Q(s, a) = Q(s, a) + \alpha\delta E(s, a)$
    $E(s, a) = \gamma\lambda E(s, a)\ \}$
    $S \leftarrow S'$; $A \leftarrow A'$
  until *meeting the terminal (target) state*;
end

Off-policy learning uses the concept of importance sampling (i.e., the estimation of the expected value of a function over a different distribution) in order to be able to learn about optimal policy while following other policies. It evaluates target policy, π, for computing $Q_\pi(s, a)$, while following a behavior policy, μ.

Off-policy TD learning uses the TD targets derived, μ to evaluate the other one, π. Then, the basic updating equation is re-written based on using one-step importance sampling correction, which is as follows:

$$V(s_t) = V(s_t) + \alpha[\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))] \tag{3.17}$$

One of the best algorithm for off-policy learning is Q-learning, which applies off-policy learning to Q-values. It is based on TD(0) and performs in a specific way without need to do importance sampling explicitly. In Q-learning, the next action, A_{t+1}, is selected based on the behavior policy, μ, while an alternative successor action based on π, A', is also considered. It means that

although the next action is actually chosen based on the behavior policy, the bootstrapping is done towards the Q-value of the alternative successor action. Therefore, the Q-value update is as follows:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[r_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)] \quad (3.18)$$

In the well-known type of Q-learning (See Algorithm) the behavior policy is, for example ε-greedy with regard to Q-values, and the target policy which we aim to improve is *greedy* with regard to Q-values. In fact, Q-learning lets both behavior and target policies improve together. Then, the target and the simplified Q-value update are as follows:

$$\begin{aligned} r_{t+1} + \gamma Q(S_{t+1}, A') &= r_{t+1} + \gamma Q(S_{t+1}, \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')) \\ &= r_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') \end{aligned} \quad (3.19)$$

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[r_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)] \quad (3.20)$$

Algorithm 3 Q-Learning Algorithm

Initialize Q-values, $\forall s \in \mathbb{S},\ \forall a \in \mathbb{A}$;
while *Not (end of learning)* **do**
    Initialize S;
    repeat
        Choose action A based on the behavior policy (e.g., using ε-greedy);
        Take action A;
        Observe S', r;
        $Q(S, A) = Q(S, A) + \alpha[r + \gamma \max_{a'} Q(S', a') - Q(S, A)]$
        $S \leftarrow S'$
    until *meeting the terminal (target) state*;
end

3.2 Performance Testing

Performance testing is a family of techniques which are intended to meet the performance evaluation objectives through executing SUT under various nor-

mal and stress execution conditions. In fact, Performance, load and stress testing are often used as interchangeable terms, even though there are also some definitions/interpretations to distinguish between them [7]. Stress testing is often considered as a type of performance testing which is intended to assess the robustness of SUT under stress conditions such as heavy workload, and/or limited resource availability to find performance breaking point. Load testing mainly addresses fulfilling performance testing objectives regarding workload-based execution conditions. Nonetheless, performance testing is mainly considered as a general term including both load and stress testing in many cases. The performance testing objectives are related to verification of performance requirements in many cases, and in case of missing performance criteria, the principle of "no worse than previous" is often used [43, 44].

Common approaches for generating performance test cases, which involve both workload-based and platform-based conditions, to measure the performance metrics and/or detecting performance issues could be categorized as follows:

System model analysis. Analysis of the performance model of SUT such as Petri nets using constraint solving techniques [45], analysis of the control flow graph of SUT using search-based techniques [46, 47], and analysis of other types of system models like UML models using genetic algorithms [16, 48, 49, 17, 18] are examples of techniques used for generating performance test cases based on system model analysis.

Source code analysis. Generating test workload based on data-flow analysis together with symbolic execution to detect performance-related issues such as exceeded response time [50, 51] are examples of using this type of approach.

Real usage modeling. Extraction of the usage pattern and modeling the real users' behavior through stochastic form-oriented models [19, 20], extraction of workload characteristics from the recorded requests using, e.g., Extended Finite State Machine (EFSM) [52] or Markov chains [53], and workload characterization using end-user clustering based on business-level attributes extracted from usage data [54] are samples of techniques for performance testing based on modeling the real users' behavior.

Behavior-driven declarative techniques. Using a Domain Specific Language (DSL) to provide declarative goal-oriented specifications of the performance testing process together with a model-driven execution framework for automated execution of the tests [22, 23, 24], and using a high-level Behavior-

Driven Load Testing (BDLT) language to provide load specification in combination with a declarative performance testing framework like BenchFlow [23, 21] are examples of declarative techniques for performance testing.

Machine learning-assisted techniques. Machine learning techniques such as supervised and unsupervised algorithms are often intended to build models and knowledge patterns from the data, while other techniques like reinforcement learning algorithms are intended to teach the agent (learner) how to solve a problem (i.e., to accomplish an objective) through interaction with the environment. Machine learning techniques have been frequently used for analyzing the resulted data from the performance testing. For example, using Bayesian Network to predict the reliability from the load testing data [55], anomaly detection based on analysis of metrics data (e.g., resource usage) using clustering techniques [56], identifying performance signature based on performance metrics data using supervised and unsupervised learning techniques [57, 58] are some examples of using machine learning techniques for analysis of performance testing data.

Machine learning techniques have also been applied to the generation of performance test conditions in some studies. For example, using RL together with symbolic execution to find the worst-case execution path within a SUT in [59], a feedback-driven learning technique which extracts some rules from the execution traces to find the performance bottlenecks, i.e., the method calls which their execution highly affects the performance of SUT [60], a feedback-based approach using search algorithms to benchmark an NFS server based on changing the test workload [61], an adaptive generation of test workload based on using pre-defined adjusting policies in [62], and using RL to find a sequence of input values resulting in performance degradation [63] are examples of using RL and RL-like techniques to generate the performance test conditions.

3.3 Performance Preservation

RL algorithms, according to their capability of finding the optimal policy to accomplish an intended objective, have been frequently used for adaptive control purposes in performance preservation of software services, such as adaptive RL-driven performance control for cloud services [64, 65, 66] and also software services on other execution platforms [37, 36]. Regarding the performance preservation techniques, in particular related to our selected application

context, real-time systems, we can classify the relevant works as runtime verification and preservation of timing requirements. Many of the verification and preservation/control approaches are based on runtime monitoring of the properties.

Real-time Specification for Java (RTSJ) [67] was originally introduced to provide a real-time scheduler for periodic threads with the facility of deadline monitoring and cost enforcement. In [68] a general model as an extension to RTSJ is introduced to support accomplishing the intended objectives of RTSJ. Ada Ravenscar Profile [69] is a strategy to preserve the timing properties of a Ravensacr compliant real-time system at runtime. It accomplishes this objective using three mechanisms as enforcing the timing properties that are to remain constant, monitoring the inherently variable timing properties, and handling the occurred violations of the properties. In [70, 71] an extra scheduler on top of a real-time operating system is proposed, which takes the timing properties including period, execution time and deadline of the tasks. It generates the real-time tasks with well-defined specification and schedules them using the underlying scheduler of the operating system to keep the temporal requirements satisfied.

There are also some studies on runtime enforcement for real-time embedded systems. For example, a runtime enforcement solution is presented in [72] which forces the system to reach an intended certain state by adding delay in the system to control the behavior of the system. It uses an offline model-based analysis to build the enforcement strategy. The solution is mainly intended to be used for testing fault tolerance-related mechanisms. A runtime reconfiguration controller, which is a command-based reconfiguration queue (CoRQ) is presented in [73]. It provides guaranteed latencies for the operations which are amenable to Worst-case Execution Time (WCET) guarantees.

Regarding monitoring and runtime verification, different types of tools have been proposed in the literature. A quick overview is as follows: A runtime model synthesis approach for timing properties of real-time systems based on monitoring the running system through using intrusive probes is proposed in [74]. The model can be used for control and verification purposes. Meanwhile, the range of existing issues in runtime monitoring of properties in real-time systems are discussed in [75] and a survey of different online and offline monitoring techniques for distributed real-time systems is presented in [76]. A runtime framework that makes an event-based model of the real-time

system for monitoring the timing properties and detecting violations of timing constraints is presented in [77]. It composes of an annotation system for specifying the timing assertions/constraints, a runtime recording and a checker for detecting violations. A tool environment with the capability of runtime monitoring, annotation, code generation, and analysis of the components to support model-driven development of real-time embedded systems is given in [78]. A runtime monitoring (verification) solution based on a feedback-based control approach with the capability of time predictability and memory utilization management is presented in [79]. art2kitekt is a runtime monitoring tool suite with a distributed architecture for tracing the temporal behavior of real-time systems [80]. GRASP [81] and Tracealyzer [82] are some other tools for tracing, visualizing and measuring different aspects of real-time embedded systems.

3.4 Performance Improvement

Regarding the performance improvement techniques in our third application context, i.e., data-intensive tasks in data grid, there are a number of studies based on using machine learning, particularly RL-based methods for resource allocation and task scheduling, of which a quick overview is as follows:

A simplified multi-agent RL is proposed for resource allocation in a grid-like environment in [83]. The agents do not interact with each other. Each agent scores the resources in terms of their efficiency. For executing a new task, the agent selects the resource with the maximum score, upon execution, it receives a reinforcement signal and updates the score of the selected resource. A dynamic resource allocation framework using RL together with a fuzzy rule base, which is called DRA-FRL, is proposed in [84]. It is intended to use RL for learning utility functions in dynamic resource allocation decisions in unknown stochastic dynamic environments with large state space.

A multi-agent learning and coordination algorithm for distributed dynamic resource allocation called Actor Critic Fuzzy Reinforcement Learning (ACFRL-2) is proposed in [85]. It extends Q-learning to the domains with large state-action space like dynamic resource allocation in grids or computer networks. Ordinal Sharing Learning (OSL) [86] is also a multi-agent reinforcement learning method for balancing the load on the nodes in large scale grids. In OSL, the agents make decisions based on shared utility tables.

In [87], the performance of SARSA is basically studied compared to queuing models on simple scenarios of resource allocation in autonomic systems. Fair Action Learning (FAL) [88] is another multi-agent learning for online resource selection in a cluster-based network. FAL finds the allocation decision policies based on a policy search technique, however it uses Q-values to approximate the policy gradients. In [89] a gradient ascent multi-agent learning called weighted policy learner (WPL) is proposed for distributed task allocation in various application domains such as grids and web services. It mainly consists of two learning problems as local resource allocation and task routing (choosing a neighbor to forward a task) problems.

A resource allocation algorithm based on using RL together with neural network is presented in [90]. The neural network component is intended to represent a resource interconnection network which is used to estimate the long-term reward. The weights of the network connections are produced based on the availability of resources. Centralized Learning Distributed Scheduling (CLDS) [91] is also a multi-agent learning approach for task scheduling in grids. It consists of two types of agents as one global learner agent and several scheduler agents. The scheduler agents shares their local rewards with the learner agent. It updates a global utility table and shares it with the scheduler agents. They use the updated utility table for resource allocation decision-making.

Chapter 4

Discussion, Conclusion and Future Work

In this chapter, we discuss our results and present a summary of conclusions, as well as a list of potential directions for future research.

4.1 Discussion and Conclusion

According to the studies in the literature, we identified room for model-free machine learning in particular model-free RL to help tackle the challenges of the involved tasks in the performance assurance process, i.e., performance testing, preservation and improvement. Modeling is a common powerful tool to accomplish the intended objectives in this area, nonetheless drawing a precise and well-detailed model which gives the status of the system given execution conditions, requires a big endeavor. Moreover, other artifacts such as source code and system models which are also used as underlying tools in many existing techniques, might not be accessible all the time. Then, we proposed that if the optimal policy (way) for accomplishing the intended objective in the performance assurance task could be learned by the arbiter system (i.e., tester, controller, or broker) instead, then the intended task could be possible without need to access source code or performance/system models. Moreover, once the optimal policy is learned, the learned policy could be reused in further situa-

tions similar to the first learning scenarios, for example, performance testing of SUTs with similar performance sensitivity to resources, which can be seen on software variants, or regression load testing scenarios of one SUT.

Therefore, the main features that lead to efficiency improvement in comparison to common approaches which are often based on ordinary search techniques are the capability of knowledge formation (in terms of Q-values and policy) during the learning, storing the gained knowledge and reusing the knowledge in further similar situations. Furthermore, the possibility of selective and adaptive control on exploration and exploitation is another strength of the proposed RL-based approaches. Regarding the involved tasks within performance assurance, a summary of the achievements is concluded as follows:

Performance testing. The proposed self-adaptive RL-driven performance testing framework learns the optimal policy to generate test cases which meet intended testing objective without access to models or source code of SUT. Once it learns, it is able to reuse the learned policy on further testing cases. For example, SaFReL, as a self-adaptive fuzzy reinforcement learning testing agent which generates performance platform-based test cases, learns how to tune the resource availability to reach the intended performance breaking point for different types of SUTs in terms of their sensitivity to resources. In other words, it learns the optimal policy to generate platform-based performance test cases resulting in reaching the intended performance breaking point for different types of SUTs, i.e., CPU-intensive, memory-intensive and disk-intensive software. Once learning the optimal policy, it replays the learned policy on further cases.

In the experimental evaluation, we aimed to address two main questions on how efficiently and adaptively SaFReL can perform on different software programs and how the efficiency of SaFReL is affected by learning parameters. We simulated performance behavior of 50 software programs of CPU-intensive, memory-intensive and disk-intensive types as SUTs running on hardware with various configurations and response time requirements (See paper B, Section 6.7). The results of the experimental evaluation show that SaFReL meets the intended objective more efficiently in comparison to a typical performance testing technique which generates performance test cases based on changing the performance test conditions, by certain steps in an exploratory way. It leads to a reduced cost in terms of computation time by reusing the learned policy upon the SUTs with similar performance sensitivity. It reuses the learned pol-

icy whenever it is useful, meanwhile, keeps the learning running in the long term to keep the learned policy updated. Table 4.1 shows the computation time improvement resulted from SaFReL on a homogeneous and heterogeneous set of SUTs (i.e., with respect to the type of performance sensitivity). A homogeneous set of SUTs refers to a set of software programs which are similar in terms of sensitivity to resources.

RELOAD as an adaptive reinforcement learning-driven load testing agent, effectively figures out the effects of different transactions involved in the workload and learns how to tune the load of transactions to meet intended testing objective (e.g., reaching a certain error rate). It learns the optimal policy to generate a workload which results in meeting the intended error rate. The intelligent tester agent can reuse the learned policy in subsequent similar testing scenarios on SUT such as regression load testing and leads to cost reduction for workload generation in further situations.

In the experimental evaluation, we tried to address two research questions on how efficient the generated test workload is and how the efficiency of RELOAD is affected by changing learning parameters. We used an e-commerce store running on a shared WordPress hosting server with one shared CPU, up to 512MB shared RAM and 100GB storage, as the SUT in the experiments (See paper C, Section 7.4). Table 4.2, which is extracted form the results, shows that RELOAD generates a more efficient workload (i.e., in terms of number of users) to meet the intended objective in comparison to a typical load testing technique. A smaller workload results in lower cost and time in the testing. A typical load testing procedure mainly involves applying a basic workload and increasing the load of the involved transactions (equally) by a certain increase step.

Performance preservation. The proposed RL-based response time control

Table 4.1: Computation time improvement in SaFReL

	SaFReL on a homogeneous set of SUTs	SaFReL on a heterogeneous set of SUTs
Action Selection Strategy: ε-greedy	$\varepsilon = 0.2$	adaptive ε
Improvement	42%	31%

Table 4.2: Efficiency of RL-driven load testing (RELOAD)

	RL-assisted with ε-greedy				
Approach	$\varepsilon = 0.85$	$\varepsilon = 0.5$	$\varepsilon = 0.2$	decaying ε	Typical load testing
Average size of generated workload	52	50	80	46	135

approach for PLC-based programs in our problem learns how to adjust delays between function blocks to keep the response time requirement satisfied. In the experimental evaluation, we tried to show how effective it can perform to preserve the response time regarding the occurrence of deviations in temporal behavior of constituent function blocks. It adapts well to the varying temporal behaviors of the function blocks while meeting the response time requirements of the programs. For example, the learning-based approach with ε-greedy, decaying ε, leads to a desired adaptation, prevents any medium or high deviations from the timing requirements, and keeps the response time requirement satisfied.

Performance improvement. The proposed learning-based resource allocation improves the performance in terms of makespan (completion time) in our application context, i.e., deployed data grid. The RL-based broker agents in the proposed approach learn how to decide about resource allocation in different situations inside the clusters to optimize the makespan (completion time) of the submitted task sets. The proposed approach with ε-greedy, $\varepsilon = 0.2$ leads to a considerable performance improvement in the experimental evaluation. The performance improvement also gets better with the increase in the size of task sets because the learned policy will be resulted from more observations and more accurate to be reused during the experiments.

Threats to Validity. A number of sources of threat to the validity of the experimental evaluation results are as follows:

Internal. The first source of threats to the results of RL-driven approaches is the formulation of the RL technique to address the problem. Modeling of the state space, formulation of actions and the reward function are major players to guide the agent throughout the learning and make it learn the optimal policy.

Another source of threats is the effect of the random selection in the action selection strategy. Reporting the average values of the measured metrics is one way to alleviate this threat during the experiments.

Finally, RL techniques like many other machine learning algorithms, are influenced by their hyperparameters such as learning rate and discount factor. During the experiments conducted for the efficiency analysis of our approaches, we did not change the learning parameters, we also conducted a set of controlled experiments to study the impact of learning parameters on the efficiency of our approaches.

External. The assumptions on the environments in our experimental evaluation experiments such as considering SUTs with combinations of three types of performance sensitivity (i.e., CPU-intensive, memory-intensive and disk-intensive) are mainly some sources of threats to validity of the results. For example, regarding the example above, if the environment contains SUTs with other types of performance sensitivity such as network-intensive programs, then the approach needs to be reformulated slightly to support new types of performance sensitivities.

4.2 Future Work

In the rest of the research journey, we are going to mainly focus on machine learning-assisted testing activities. This thesis identifies room for some research directions as future work, which are outlined as follows:

In major cases in this thesis, the use of simulation-based evaluation is intended to show the applicability of RL-assisted approaches to the performance assurance fields and how the involved principles of the approaches (the ideas and heuristics) work. Meanwhile, there are also some heuristics and customized techniques for RL techniques to facilitate the use of them in realistic scaled-up environments with more complexity. Therefore, using scaling up techniques such as using different function approximations instead of Q-tables to deal with the problems with large MDPs and improving the learning through using some heuristics and techniques to speed up the exploration of state space such as applying multi-agent learning techniques could be some future directions with respect to the learning technique.

Analyzing the results of the performance testing to find out the causes of

anomalies in the performance behavior of the system, i.e., performance bottlenecks, could be another opportunity for future work.

The reduction of dependency on system model and source code is one of the strengths of applying model-free RL algorithms, particularly for accomplishing the testing objectives on complex systems. Machine learning-enabled systems such as deep neural network-based systems, are examples of complex systems for which drawing a model of their behavior is difficult. Thus there will be interesting room for using RL-assisted testing techniques, in particular for finding adversarial test cases, which result in abnormal behavior after a slight and minor perturbation. Regarding the incorporation of machine learning components into many safety-critical applications, testing of ML-enabled systems with regard to safety requirements is of great importance. Therefore, RL-assisted testing of ML systems, in particular DNNs, could be an interesting future direction.

Bibliography

[1] Leonid Grinshpan. *Solving enterprise applications performance puzzles: queuing models to the rescue*. John Wiley & Sons, 2012.

[2] Andreas Brunnert, André van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Herbst, Pooyan Jamshidi, Reiner Jung, Joakim von Kistowski, et al. Performance-oriented devops: A research agenda. *arXiv preprint arXiv:1508.04752*, 2015.

[3] Elaine J Weyuker and Filippos I Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE transactions on software engineering*, 26(12):1147–1156, 2000.

[4] ISO 25000. ISO/IEC 25010 - System and software quality models, 2019. Available at https://iso25000.com/index.php/en/iso-25000-standards/iso-25010, Retrieved July, 2019.

[5] Martin Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26. IEEE, 2007.

[6] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.

[7] Zhen Ming Jiang and Ahmed E Hassan. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, 41(11):1091–1118, 2015.

[8] Dorina Petriu, Christiane Shousha, and Anant Jalnapurkar. Architecture-based performance analysis applied to a telecommunication system. *IEEE Transactions on Software Engineering*, 26(11):1049–1065, 2000.

[9] Simona Bernardi, Susanna Donatelli, and José Merseguer. From uml sequence diagrams and statecharts to analysable petri net models. In *Proceedings of the 3rd international workshop on Software and performance*, pages 35–45. ACM, 2002.

[10] Edward D Lazowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., 1984.

[11] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-based software performance analysis*. Springer Science & Business Media, 2011.

[12] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[13] Krishna Kant and MM Srinivasan. *Introduction to computer system performance evaluation*. McGraw-Hill College, 1992.

[14] Giovanni Denaro, Andrea Polini, and Wolfgang Emmerich. Early performance testing of distributed software applications. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 94–103. ACM, 2004.

[15] Pingyu Zhang, Sebastian Elbaum, and Matthew B Dwyer. Compositional load test generation for software pipelines. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 89–99. ACM, 2012.

[16] Vahid Garousi. A genetic algorithm-based stress test requirements generator tool and its empirical evaluation. *IEEE Transactions on Software Engineering*, 36(6):778–797, 2010.

[17] Leandro T Costa, Ricardo M Czekster, Flávio Moreira de Oliveira, Elder de M Rodrigues, Maicon Bernardino da Silveira, and Avelino F Zorzo. Generating performance test scripts and scenarios based on abstract intermediate models. In *SEKE*, pages 112–117, 2012.

[18] Maicon Bernardino da Silveira, Elder de M Rodrigues, Avelino F Zorzo, Leandro T Costa, Hugo V Vieira, and Flávio Moreira de Oliveira. Generation of scripts for performance testing based on uml models. In *SEKE*, pages 258–263, 2011.

[19] Dirk Draheim, John Grundy, John Hosking, Christof Lutteroth, and Gerald Weber. Realistic load testing of web applications. In *Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 11–pp. IEEE, 2006.

[20] Christof Lutteroth and Gerald Weber. Modeling a realistic workload for performance testing. In *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 149–158. IEEE, 2008.

[21] Henning Schulz, Dušan Okanović, André van Hoorn, Vincenzo Ferme, and Cesare Pautasso. Behavior-driven load testing using contextual knowledge-approach and experiences. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pages 265–272. ACM, 2019.

[22] Vincenzo Ferme and Cesare Pautasso. A declarative approach for performance tests execution in continuous software development environments. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 261–272. ACM, 2018.

[23] Vincenzo Ferme and Cesare Pautasso. Towards holistic continuous software performance assessment. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, pages 159–164. ACM, 2017.

[24] Jürgen Walter, Andre van Hoorn, Heiko Koziolek, Dusan Okanovic, and Samuel Kounev. Asking what?, automating the how?: The vision of declarative performance engineering. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, pages 91–94. ACM, 2016.

[25] Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. Performance anomaly detection and bottleneck identification. *ACM Computing Surveys (CSUR)*, 48(1):4, 2015.

[26] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[27] Brendan Gregg. *Systems performance: enterprise and the cloud*. Pearson Education, 2013.

[28] Gordana Dodig-Crnkovic. Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*, pages 126–130, 2002.

[29] Hilary J Holz, Anne Applin, Bruria Haberman, Donald Joyce, Helen Purchase, and Catherine Reed. Research methods in computing: what are they, and how should we teach them? *ACM SIGCSE Bulletin*, 38(4):96–114, 2006.

[30] Victor Basili, Lionel Briand, Domenico Bianculli, Shiva Nejati, Fabrizio Pastore, and Mehrdad Sabetzadeh. Software engineering research and industry: a symbiotic relationship to foster impact. *IEEE Software*, 35(5):44–49, 2018.

[31] Colin Robson and Kieran McCartan. *Real world research*. John Wiley & Sons, 2016.

[32] Forrest Shull, Janice Singer, and Dag IK Sjøberg. *Guide to advanced empirical software engineering*. Springer, 2007.

[33] Robert Feldt. Guide to research questions, 2010. Available at `http://www.robertfeldt.net/advice/index.html`, Retrieved March, 2020.

[34] Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. Machine learning to guide performance testing: An autonomous test framework. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 164–167. IEEE, 2019.

[35] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[36] Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. Adaptive runtime response time control in plc-based real-time systems using reinforcement learning. In *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 217–223. IEEE, 2018.

[37] Mahshid Helali Moghadam and Seyed Morteza Babamir. Makespan reduction for dynamic workloads in cluster-based data grids using reinforcement-learning based scheduling. *Journal of computational science*, 24:402–412, 2018.

[38] Taiwo Oladipupo Ayodele. Types of machine learning algorithms. *New advances in machine learning*, pages 19–48, 2010.

[39] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[40] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[41] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[42] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[43] Frank Huebner, Kathleen Meier-Hellstern, and Paul Reeser. Performance testing for ip services and systems. In *Performance Engineering*, pages 283–299. Springer, 2000.

[44] Ivo Jimenez, Noah Watkins, Michael Sevilla, Jay Lofstead, and Carlos Maltzahn. Quiho: Automated performance regression testing using inferred resource utilization profiles. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 273–284. ACM, 2018.

[45] Jian Zhang and Shing Chi Cheung. Automated test case generation for the stress testing of multimedia systems. *Software: Practice and Experience*, 32(15):1411–1435, 2002.

[46] Yuanyan Gu and Yujia Ge. Search-based performance testing of applications with composite services. In *2009 International Conference on Web Information Systems and Mining*, pages 320–324. IEEE, 2009.

[47] Massimiliano Di Penta, Gerardo Canfora, Gianpiero Esposito, Valentina Mazza, and Marcello Bruno. Search-based testing of service level agreements. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1090–1097. ACM, 2007.

[48] Vahid Garousi. Empirical analysis of a genetic algorithm-based stress test technique. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1743–1750. ACM, 2008.

[49] Vahid Garousi, Lionel C Briand, and Yvan Labiche. Traffic-aware stress testing of distributed real-time systems based on uml models using genetic algorithms. *Journal of Systems and Software*, 81(2):161–185, 2008.

[50] Cheer-Sun D Yang and Lori L Pollock. Towards a structural load testing tool. In *ACM SIGSOFT Software Engineering Notes*, volume 21, pages 201–208. ACM, 1996.

[51] Pingyu Zhang, Sebastian Elbaum, and Matthew B Dwyer. Automatic generation of load tests. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 43–52. IEEE Computer Society, 2011.

[52] Mahnaz Shams, Diwakar Krishnamurthy, and Behrouz Far. A model-based approach for testing the performance of web applications. In *Proceedings of the 3rd international workshop on Software quality assurance*, pages 54–61. ACM, 2006.

[53] Christian Vögele, André van Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. Wessbas: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems. *Software & Systems Modeling*, 17(2):443–477, 2018.

[54] Gururaj Maddodi, Slinger Jansen, and Rolf de Jong. Generating workload for erp applications through end-user organization categorization using high level business operation data. In *Proceedings of the*

2018 ACM/SPEC International Conference on Performance Engineering, pages 200–210. ACM, 2018.

[55] Alberto Avritzer, Flávio P Duarte, Rosa Maria Meri Leao, Edmundo de Souza e Silva, Michal Cohen, and David Costello. Reliability estimation for large distributed software systems. In *Cascon*, page 12. Citeseer, 2008.

[56] Mark D Syer, Bram Adams, and Ahmed E Hassan. Identifying performance deviations in thread pools. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 83–92. IEEE, 2011.

[57] Haroon Malik, Hadi Hemmati, and Ahmed E Hassan. Automatic detection of performance deviations in the load testing of large scale systems. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1012–1021. IEEE Press, 2013.

[58] Haroon Malik, Zhen Ming Jiang, Bram Adams, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. Automatic comparison of load tests to support the performance analysis of large enterprise systems. In *2010 14th European conference on software maintenance and reengineering*, pages 222–231. IEEE, 2010.

[59] Jinkyu Koo, Charitha Saumya, Milind Kulkarni, and Saurabh Bagchi. Pyse: Automatic worst-case test generation by reinforcement learning. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 136–147. IEEE, 2019.

[60] Mark Grechanik, Chen Fu, and Qing Xie. Automatically finding performance problems with feedback-directed learning software testing. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 156–166. IEEE, 2012.

[61] Piyush Shivam, Varun Marupadi, Jeffrey S Chase, Thileepan Subramaniam, and Shivnath Babu. Cutting corners: Workbench automation for server benchmarking. In *USENIX Annual Technical Conference*, pages 241–254, 2008.

[62] Vanessa Ayala-Rivera, Maciej Kaczmarski, John Murphy, Amarendra Darisa, and A Omar Portillo-Dominguez. One size does not fit all: In-test

workload adaptation for performance testing of enterprise applications. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 211–222. ACM, 2018.

[63] Tanwir Ahmad, Adnan Ashraf, Dragos Truscan, and Ivan Porres. Exploratory performance testing using reinforcement learning. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 156–163. IEEE, 2019.

[64] Olumuyiwa Ibidunmoye, Mahshid Helali Moghadam, Ewnetu Bayuh Lakew, and Erik Elmroth. Adaptive service performance control using cooperative fuzzy reinforcement learning in virtualized environments. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 19–28. ACM, 2017.

[65] T Veni and S Mary Saira Bhanu. Auto-scale: automatic scaling of virtualised resources using neuro-fuzzy reinforcement learning approach. *International Journal of Big Data Intelligence*, 3(3):145–153, 2016.

[66] Pooyan Jamshidi, Amir Sharifloo, Claus Pahl, Hamid Arabnejad, Andreas Metzger, and Giovani Estrada. Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. In *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*, pages 70–79. IEEE, 2016.

[67] Gregory Bollella and James Gosling. The real-time specification for java. *Computer*, 33(6):47–54, 2000.

[68] Andy Wellings, Gregory Bollella, Peter Dibble, and David Holmes. Cost enforcement and deadline monitoring in the real-time specification for java. In *Seventh IEEE International Symposium onObject-Oriented Real-Time Distributed Computing, 2004. Proceedings.*, pages 78–85. IEEE, 2004.

[69] Enrico Mezzetti, Marco Panunzio, and Tullio Vardanega. Preservation of timing properties with the ada ravenscar profile. In *International Conference on Reliable Software Technologies*, pages 153–166. Springer, 2010.

[70] Mehrdad Saadatmand, Mikael Sjödin, and Naveed Ul Mustafa. Monitoring capabilities of schedulers in model-driven development of real-time

systems. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pages 1–10. IEEE, 2012.

[71] Nima Asadi, Mehrdad Saadatmand, and Mikael Sjödin. Run-time monitoring of timing constraints: A survey of methods and tools. In *the Eighth International Conference on Software Engineering Advances*, 2013.

[72] Louis-Marie Givel, Jean-Luc Béchennec, Matthias Brun, Sébastien Faucou, and Olivier H Roux. Testing real-time embedded software using runtime enforcement. In *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–6. IEEE, 2016.

[73] Marvin Damschen, Lars Bauer, and Jörg Henkel. Corq: Enabling runtime reconfiguration under wcet guarantees for real-time systems. *IEEE Embedded Systems Letters*, 9(3):77–80, 2017.

[74] Joel Huselius and Johan Andersson. Model synthesis for real-time systems. In *Ninth European Conference on Software Maintenance and Reengineering*, pages 52–60. IEEE, 2005.

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

[76] Alwyn E Goodloe and Lee Pike. Monitoring distributed real-time systems: A survey and future directions. 2010.

[77] Farnam Jahanian. Run-time monitoring of real-time systems. *Advances in Real-Time Systems. Prentice Hall*, 1995.

[78] Nondini Das, Suchita Ganesan, Leo Jweda, Mojtaba Bagherzadeh, Nicolas Hili, and Juergen Dingel. Supporting the model-driven development of real-time embedded systems with run-time monitoring and animation via highly customizable code generation. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 36–43. ACM, 2016.

[79] Ramy Medhat, Borzoo Bonakdarpour, Deepak Kumar, and Sebastian Fischmeister. Runtime monitoring of cyber-physical systems under timing

and memory constraints. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(4):79, 2015.

[80] Miguel García-Gordillo, Joan J Valls, and Sergio Sáez. Heterogeneous runtime monitoring for real-time systems with art2kitekt. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 266–273. IEEE, 2019.

[81] Mike Holenderski, MMHP Van Den Heuvel, Reinder J Bril, and Johan J Lukkien. Grasp: Tracing, visualizing and measuring the behavior of real-time systems. In *International workshop on analysis tools and methodologies for embedded and real-time systems (WATERS)*, pages 37–42, 2010.

[82] Johan Kraft, Anders Wall, and Holger Kienle. Trace recording for embedded systems: Lessons learned from five industrial projects. In *International Conference on Runtime Verification*, pages 315–329. Springer, 2010.

[83] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource allocation in the grid with learning agents. *Journal of Grid Computing*, 3(1-2):91–100, 2005.

[84] David Vengerov. A reinforcement learning approach to dynamic resource allocation. 2005.

[85] David Vengerov. Multi-agent learning and coordination algorithms for distributed dynamic resource allocation. 2004.

[86] Jun Wu, Xin Xu, Pengcheng Zhang, and Chunming Liu. A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Generation Computer Systems*, 27(5):430–439, 2011.

[87] Gerald Tesauro, Rajarshi Das, William E Walsh, and Jeffrey O Kephart. Utility-function-driven resource allocation in autonomic systems. In *Second International Conference on Autonomic Computing (ICAC'05)*, pages 342–343. IEEE, 2005.

[88] Chongjie Zhang, Victor Lesser, and Prashant Shenoy. A multi-agent learning approach to online distributed resource allocation. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[89] Sherief Abdallah and Victor Lesser. Learning the task allocation game. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 850–857. ACM, 2006.

[90] Masnida Hussin, Nor Asilah Wati Abdul Hamid, and Khairul Azhar Kasmiran. Improving reliability in resource management through adaptive reinforcement learning for distributed systems. *Journal of parallel and distributed computing*, 75:93–100, 2015.

[91] Milad Moradi. A centralized reinforcement learning method for multi-agent job scheduling in grid. In *2016 6th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 171–176. IEEE, 2016.

II

Included Papers

Chapter 5

Paper A: Machine Learning to Guide Performance Testing: An Autonomous Test Framework

Mahshid Helali Moghadam, Mehrdad Saadatmand,
Markus Borg, Markus Bohlin, Björn Lisper
In the Proceedings of the 12th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). Xian, China, April 2019.

Abstract

Satisfying performance requirements is of great importance for performance-critical software systems. Performance analysis to provide an estimation of performance indices and ascertain whether the requirements are met is essential for achieving this target. Model-based analysis as a common approach might provide useful information but inferring a precise performance model is challenging, especially for complex systems. Performance testing is considered as a dynamic approach for doing performance analysis. In this work-in-progress paper, we propose a self-adaptive learning-based test framework which learns how to apply stress testing as one aspect of performance testing on various software systems to find the performance breaking point. It learns the optimal policy of generating stress test cases for different types of software systems, then replays the learned policy to generate the test cases with less required effort. Our study indicates that the proposed learning-based framework could be applied to different types of software systems and guides towards autonomous performance testing.

5.1 Introduction

Nowadays, on one hand we face the increasing number of software-based services, on the other hand, the expectations on the quality of these services are considerably rising. In general, the properties of a software system could be described in terms of functional and non-functional aspects. Non-functional properties describe the quality of functional aspects of the system and represent quality attributes like *performance* [1]. Requirements are the main means against which the functional and non-functional aspects of a system are assessed. The non-functional requirements are often described based on the software metrics quantifying the non-functional properties of the system. Assessing the satisfaction of non-functional requirements plays a crucial role in assuring the user's expected quality and even the behavioral correctness in many systems, particularly resource-constrained, and safety critical systems.

Performance is a non-functional property indicating the operational efficiency of a software system with respect to different execution conditions like various types of workload and allocation of available resources [2]. It is measured and quantified using multiple indices such as response time, throughput, and resource utilization. Performance analysis could be done through both performance modeling and performance testing. Performance modeling generally involves identifying the proper performance indices, building a performance model expressing the relevant indices. Consequently, different model-driven engineering techniques like model verification, model refactoring, and performance tuning could be performed based on the performance model.

Performance testing is intended to ascertain whether the software system performs well under the actual execution conditions (i.e., internal and external factors affecting the performance) and meets the performance requirements. Various methods have been proposed for building software performance models [3, 4, 5, 6]. Performance models might provide helpful hints of the performance of the system and even probable bottlenecks in the architecture; however, they cannot represent the whole details of the system. For example, many of the details of the deployment environment may be ignored in the models [7], although they might still have significant effects on the performance of the system. Testing the software system under stress, which is called stress testing, is one of directions involved in performance testing. The main objective is to find the performance breaking point, at which the systems breaks, or performance

requirements are not met anymore. Two general views as internal and external could be assigned to performance analysis. Analysis with internal view considers internal conditions causing a performance bottleneck and consequently affecting the performance of the system. Performance testing with external view is evaluating/examining how the system will perform under different external conditions like heavier workload and limited resource availability.

In this paper, we present a learning-based self-adaptive framework for providing autonomous performance testing. The proposed smart framework is able to learn how to apply stress testing efficiently to different types of software systems, including CPU-intensive, memory-intensive and disk-intensive programs, to find the performance breaking point. It basically uses model-free reinforcement learning (RL), i.e., Q-learning with multiple experience (knowledge) bases to learn the policy for finding performance breaking point of different types of software under test (SUT) without having performance models.

The rest of this paper is organized as follows; Section 5.2 discusses the background concepts of RL and the motivation for proposing learning-based testing, Section 5.3 presents the details of the proposed smart performance testing framework, with a short discussion on its applicability and operational performance. Section 5.4 provides a review on the background relevant approaches. The paper concludes with a conclusion and future directions of this work-in-progress research in Section 5.5.

5.2 Motivation and Background

Performance analysis is an essential step towards performance assurance to keep the performance requirements satisfied. Performance testing and performance modeling are dynamic and static approaches for realizing performance analysis. Regarding complex systems, providing a precise model of the system and execution environment is challenging. In the context of performance testing, the complexity of SUTs and the dynamism of the performance affecting factors in execution environments are the major barriers motivating application of learning-based performance testing.

Reinforcement learning [8] has been frequently used as one of the key approaches for building self-adaptive smart systems. RL is a semi-supervised learning involving interaction with the environment. In RL, an agent (the learner) continuously detects the status (state) of the environment (the system

under control). Then, it selects an action to be applied on the environment and in return it receives a reinforcement signal (reward signal) showing the effectiveness of the action. The final objective during the learning, is to find a policy maximizing the total long-term received reward. The agent mainly uses a strategy based on a combination of trying out actions (exploration) and selecting highly valued actions (exploitation). Q-learning [8] is a well-known model-free algorithm in the context of RL, in which the agent learns the utility value of the long-term reward associated to pairs of states and actions. Q-learning is off-policy, since the agent learns the optimal policy independently of the selected strategy for the action selection step.

5.3 Self-Adaptive Learning-Based Performance Testing

This section presents the architecture and operating procedure of a smart framework providing autonomous performance testing. It focuses on stress testing as one of the main target fields in the scope of performance testing. It supports automated performance test case generation for different software systems without having performance models. The proposed framework as a smart agent uses reinforcement learning as its core learning mechanism. It aims at learning how to find the performance breaking point of various software systems depending on their performance sensitivity nature. The learning mechanism includes *initial convergence* and *transfer learning phases*.

Initial convergence. An initial experience (knowledge) convergence is achieved upon the first learning episodes in interaction with the first SUT instance of each type. The smart agent stores the achieved experience under three experience (knowledge) bases, i.e., experience for CPU-intensive, memory-intensive and disk-intensive SUTs. Therefore, the experience bases initially converge upon interaction with the first CPU-intensive, memory-intensive and disk-intensive SUT respectively.

Transfer learning. After the initial convergence of experience bases, the smart agent keeps the learning running to update the knowledge bases upon observing new SUTs. It is supposed that during the transfer learning, the smart agent mostly relies on the achieved experience, while also partly explores the environment to keep the gained knowledge updated. Using the learnt policy

during the interaction with SUT instances, causes the agent to generate the stress test cases/test conditions to find the performance breaking point with less effort (in terms of learning trials) and leads to a better efficiency. Experience exploitation is the key concept of this phase which results in more efficiency in test case generation. The policies learnt for CPU-, memory- and disk-intensive programs are quite different. Then, this is where separating the experience bases of the agent is beneficial. Upon observing a CPU-, memory- or disk-intensive SUT, the agent activates the corresponding experience base for taking actions on the observed SUT instance. Figure 5.1 depicts an overview of the architecture of our smart tester agent. The details including the components, and main steps of the learning part is as follows:

I. State Detection. Detecting the current state of the system is one of the main steps of an RL-based algorithm. In our smart framework, four measurements of the SUT and execution environment including CPU, memory and disk utilization, and also SUT response time are used to specify the state of the system. The state detector component receives a tuple consisting of (CPU_U, Mem_U, $Disk\_U$, Rt) as input to specify the state of the system, where CPU_U, Mem_U, $Desk\_U$, Rt present the CPU, memory, disk utilization and response time respectively. These continuous parameters form the state space of the system, then the next step is dividing the state space into multiple discrete states. The values of these parameters are classified into multiple classes to specify the discrete states of the system, as shown in Figure 5.2.

II. Apply Action. After state detection, the agent applies one possible action to the system. Actions are operations which change (reduce) the available resources including CPU cores, memory and disk, and also change the factors affecting the performance like increasing the workload. In the first prototype of our smart framework, actions include the operations modifying the available resources by a decreasing factor:

$$DecFac\_CPU = \left\{\tfrac{1}{4}, \tfrac{2}{4}, \tfrac{3}{4}, 1\right\} \tag{5.1}$$

$$DecFac\_MemDisk = \left\{d.\tfrac{memory(disk)}{4} \mid d \in \left\{\tfrac{1}{4}, \tfrac{2}{4}, \tfrac{3}{4}, 1\right\}\right\} \tag{5.2}$$

where memory ($disk$) represent the current amount of available memory (disk).

Then, the set of actions have been specified as shown in Table 5.1.

III. Compute Reward. After applying each action, the agent computes a reward signal showing the effectiveness of the applied action. The reward is calculated using the following utility function:

$$U(n) = kU_r(n) + (1-k)U_E \tag{5.3}$$

where $U_r(n)$ indicates to what extent the response time of the system deviates from the acceptable region, U_E represents the efficiency of the resource usage, and $k, 0 \leq k \leq 1$ is a weighting parameter to allow the agent to prioritize different aspects of the stress conditions.

IV. Experience Adaptation. This component receives a performance sensitivity indication expressing the type of sensitivity of SUT, i.e., being CPU-, memory- or disk-intensive. Then, it selects the corresponding experience (knowledge) base for the stress test case generation on the observed SUT.

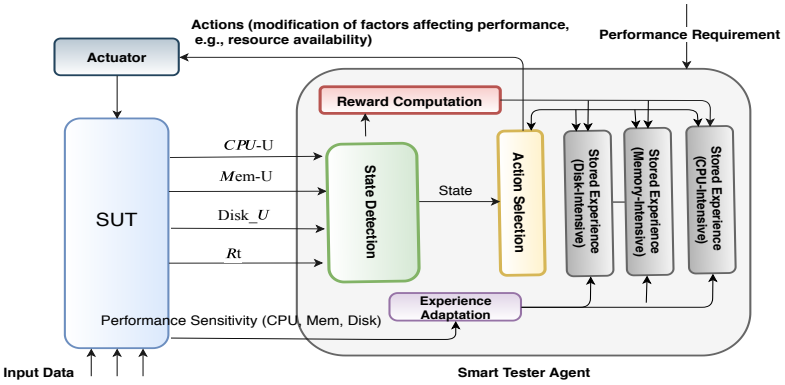


Figure 5.1: Architecture of the smart framework

How the learning works. The concept of the achieved experience in RL is defined in terms of policy. A policy is defined as a mapping between the states and actions and specifies the action which should be taken in each state. A utility value, $Q^{\pi}(s,a)$, is assigned to taking action, a, in given state s, according to the policy π. $Q^{\pi}(s,a)$ as the expected long-term reward of the pair (s,a) is

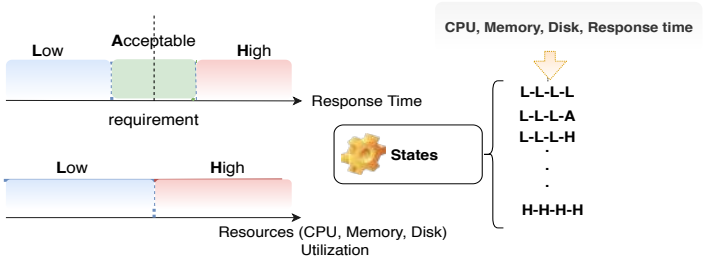


Figure 5.2: States of the system

Table 5.1: set of actions

Actions	
Reducing the available CPU cores	by a factor in $DecFac\_CPU$
Reducing the available memory	by a factor in $DecFac\_MemDisk$
Reducing the available disk	
No action	

defined as follows [8]:

$$Q^{\pi}(s,a) = E^{\pi}[R_n|S_n = s, A_n = a] \tag{5.4}$$

$$R_n = \sum_{k=0}^{\infty} \gamma^k r_{n+k+1} \tag{5.5}$$

Where S_n ,A_n and $r_{(n+k+1)}$ are current state, action and future rewards respectively. $\gamma \in [0,1]$ is a discount factor specifying to what extent the agent gives more weight to the future reward compared to the immediately achieved reward. Q-values are stored in a look-up table (Q-table) and considered as the experience of the agent. The Q-values are used for deciding between actions when the agent relies on using its experience (exploitation). The Q-values are also updated incrementally (via temporal differencing) during the learning using Eq. 5.6. The final objective of Q-learning is finding a policy maximizing the expected long-term reward of pairs of states and actions.

$$Q(s_n,a_n) = Q(s_n,a_n) + \alpha[r_{n+1} + \gamma \max_{a'} Q(s_{n+1},a') - Q(s_n,a_n)] \tag{5.6}$$

Learning performance. Different methods like ε-greedy with various ε-values and Softmax can be used for action selection. They provide different trade-offs between exploration and exploitation of the state-action space which could impact the efficiency of the learning e.g., in terms of convergence speed. Setting different values for learning parameters such as discount factor γ, and learning rate α could also affect the learning performance.

How the performance test is done. In our smart framework, the agent learns how to provide efficiently stress conditions for different types of SUTs to find the performance breaking point, from which the performance requirement of the SUT is not met anymore. The agent stops applying actions upon reaching the breaking point. Algorithm 4 presents the operating procedure of our learning-based performance testing framework.

Applicability. Performance-critical programs are sensitive parts in many software-intensive systems like industrial control systems. The proposed framework provides a model-free autonomous performance testing which could be easily applied to different types of software-intensive systems. Virtualized test environments would be a perfect infrastructure for executing this approach. Moreover, the proposed framework would be also integrated into the simulation environments which could be highly useful for testing purposes on industrial software systems.

5.4 Related Work

Model-based software performance engineering is mainly based on building a performance model of the system. Some of the specific modeling notations used for performance modeling are Queueing Networks, Markov Process, Petri Nets, Process Algebras and also simulation models [3, 4, 5, 6]. Pushing towards automation in generating the performance model is essential to eliminate the manual model generation and bring the performance analysis to early stages in the software life cycle. There has been a substantial literature published in the area of software performance modeling [2, 9, 10]. In the context of testing, although performance testing tightly overlaps with performance modeling in some cases, it is intended to satisfy some partially different objectives. Performance testing, load testing and stress testing are three terms which are used in many cases interchangeably [11].

Algorithm 4 Learning-based Performance Testing

Initialize Q-values, $Q(s, a) = 0 \forall\ s\ \in\ S, \forall\ a \in\ A$
Observe and identify the type of the SUT instance.
if SUT is the first instance of CPU-intensive, memory-intensive or disk-intensive instances {
repeat
1. Detect the state of the SUT
2. Select an action based on the action selection strategy
3. Apply the selected action, re-execute the SUT
4. Detect the new state of the system regarding the selected action.
5. Receive the reward signal, r_{n+1}
6. Update the Q-value in the corresponding experience base (Q-table)

until *the initial convergence* `/* Initial Convergence */`;
} **Else**{
7. Select the proper experience (knowledge) base
repeat
8. Detect the state of the SUT
9. Select an action,
$a_n = argmax_{a \in A} Q(s_n, a)$ from the experience base with probability (1- ε) or a random action with probability $\varepsilon, \varepsilon \leq 0.2$)
10. Apply the selected action, re-execute the SUT
11. Detect the new state of the system
12. Receive the reward signal, r_{n+1}
13. Update the Q-value
until *finding performance breaking point* `/* Transfer learning */`;
}

In general, load testing has been considered as behavior assessment of a SUT under load expected in a real-world execution context, from two perspectives of functional problems and violation of non-functional requirements caused under load. Stress testing has been defined as the behavioral assessment of a SUT under extreme conditions including heavy load and limited available resources [11]. Performance testing is often considered as a more general term which often includes both load testing and stress testing. There are many commonalities between these types of testing. Regarding common

and different performance test scenarios, the behavioral assessment of a SUT from the perspective of performance-related issues and aspects generally aim at the following objectives:
I. Measurement of performance under load and/or different resource configuration. This process might overlap with performance modeling in many cases. It can be done under expected load or stress conditions [12, 13, 14, 15].
II. Detection of functional problems under load and/or different resource configuration. It can be also done under expected load or stress conditions [16, 17].
III. Detection of performance requirements violation such as violating reliability, and robustness requirements. This process can also be done under typical expected load or stress conditions [18].

This work-in-progress paper proposes a learning-based framework for performance testing, in particular stress testing.

5.5 Conclusion

Performance analysis to provide an estimation of performance indices in different execution conditions is a challenge for complex software systems. In addition to static model-driven techniques, performance testing is considered as a dynamic approach for performance analysis. Efficient automated test case/test condition generation is a challenging activity in software testing. In this paper, we present a self-adaptive learning-based framework to conduct stress testing on various software systems without having the performance models of the systems. We used Q-learning as a model-free RL algorithm in our smart test framework. It learns the optimal policy of generating stress test cases for various software systems. After the initial learning convergence, it uses the learnt policy for further SUT instances and generates test cases efficiently with less required effort. Detailed efficacy analysis of the proposed framework on different software systems and deploying it on a virtualized infrastructure will be our next steps in this research.

Acknowledgements

This research has been funded partially by Vinnova through the ITEA3 TESTOMAT (www.testomatproject.eu) and XIVT.

Bibliography

[1] Martin Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26. IEEE, 2007.

[2] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-based software performance analysis*. Springer Science & Business Media, 2011.

[3] Dorina Petriu, Christiane Shousha, and Anant Jalnapurkar. Architecture-based performance analysis applied to a telecommunication system. *IEEE Transactions on Software Engineering*, 26(11):1049–1065, 2000.

[4] Rob Pooley. Using uml to derive stochastic process algebra models. 1999.

[5] Simona Bernardi, Susanna Donatelli, and José Merseguer. From uml sequence diagrams and statecharts to analysable petri net models. In *Proceedings of the 3rd international workshop on Software and performance*, pages 35–45, 2002.

[6] Edward D Lazowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., 1984.

[7] Giovanni Denaro, Andrea Polini, and Wolfgang Emmerich. Early performance testing of distributed software applications. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 94–103. ACM, 2004.

[8] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[9] Mike Kosta Loukides. *System performance tuning*. O'Reilly & Associates, Inc., 1996.

[10] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[11] Zhen Ming Jiang and Ahmed E Hassan. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, 41(11):1091–1118, 2015.

[12] Daniel A Menascé. Load testing, benchmarking, and application performance management for the web. In *Int. CMG Conference*, pages 271–282, 2002.

[13] Mitashree Kalita and Tulshi Bezboruah. Investigation on performance testing and evaluation of prewebd: A. net technique for implementing web application. *IET software*, 5(4):357–365, 2011.

[14] Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. Learning-based response time analysis in real-time embedded systems: a simulation-based approach. In *Proceedings of the 1st International Workshop on Software Qualities and Their Dependencies*, pages 21–24. ACM, 2018.

[15] Olumuyiwa Ibidunmoye, Mahshid Helali Moghadam, Ewnetu Bayuh Lakew, and Erik Elmroth. Adaptive service performance control using cooperative fuzzy reinforcement learning in virtualized environments. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 19–28. ACM, 2017.

[16] Bruno Dillenseger. Clif, a framework based on fractal for flexible, distributed load testing. *annals of telecommunications-annales des télécommunications*, 64(1-2):101–120, 2009.

[17] Frank Huebner, Kathleen Meier-Hellstern, and Paul Reeser. Performance testing for ip services and systems. In *Performance Engineering*, pages 283–299. Springer, 2000.

[18] Saeed Abu-Nimeh, Suku Nair, and Marco Marchetti. Avoiding denial of service via stress testing. In *IEEE International Conference on Computer Systems and Applications, 2006.*, pages 300–307. IEEE, 2006.

Chapter 6

Paper B: An Autonomous Performance Testing Framework Using Self-Adaptive Fuzzy Reinforcement Learning

Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, Björn Lisper
Submitted to a Special Issue on Testing of Software and Systems, Software Quality Journal, February 2020.

Abstract

Test automation brings the potential to reduce costs and human effort, but several aspects of software testing remain challenging to automate. One such example is automated performance testing to find performance breaking points. Current approaches to tackle automated generation of performance test cases mainly involve using source code or system model analysis or use-case based techniques. However, source code and system models might not always be available at testing time. On the other hand, if the optimal performance testing policy for the intended objective in a testing process instead could be learned by the testing system, then test automation without advanced performance models could be possible. Furthermore, the learned policy could later be reused for similar software systems under test, thus leading to higher test efficiency. We propose SaFReL, a self-adaptive fuzzy reinforcement learning-based performance testing framework. SaFReL learns the optimal policy to generate performance test cases through an initial learning phase, then reuses it during a transfer learning phase, while keeping the learning running and updating the policy in the long term. Through multiple experiments on a simulated environment, we demonstrate that our approach generates the target performance test cases for different programs more efficiently than a typical testing process, and performs adaptively without access to source code and performance models.

6.1 Introduction

Quality assurance with respect to both functional and non-functional quality characteristics of software becomes crucial to success of software products. For example, an extra one second delay in load time of a storefront page can cause 11% reduction in page views, and 16% less customer satisfaction [1]. Moreover, banking, retailing and airline reservation systems as samples of mission-critical systems are all required to be resilient against varying conditions affecting their functional performance [2, 3, 4].

Performance which has been also called "efficiency" in the classification schemes of quality characteristics [5, 6, 7], is generally referred to as how well a software system (service) accomplishes the expected functionalities. Performance requirements mainly describe time and resource bound constraints on the behavior of software, which are often expressed in terms of performance metrics such as response time, throughput, and resource utilization.

Performance evaluation. Performance modeling and testing are common evaluation approaches to accomplish the associated objectives such as measurement of performance metrics, detection of functional problems emerging under certain performance conditions, and also violations of performance requirements [8]. Performance modeling mainly involves building a model of the software system's behavior using modeling notations such as queueing networks, Markov processes, Petri nets and simulation models [9, 10, 11]. Although models provide helpful insight of the performance behavior of the system, there are also many details of implementation and execution environment that might be ignored in the modeling [12]. Moreover, drawing a precise model expressing the performance behavior of the software under different conditions is often difficult. Performance testing as another family of techniques is intended to achieve the aforementioned objectives through executing the software under the actual conditions.

Verifying robustness of the system in terms of finding performance breaking point is one of the primary purposes of performance testing. A performance breaking point refers to a status of software at which the system becomes unresponsive or certain performance requirements get violated.

Research challenge. Performance testing to find performance breaking points, remains a challenge for complex software and execution environments. Testing approaches mainly raise issues of automated and efficient generation

of test cases (test conditions) resulting in accomplishing the intended objective. Common approaches for generating the performance test cases such as using source code analysis [13], linear programs and evolutionary algorithms on performance models [14, 15, 16] and UML models [17, 18, 19, 20, 21], using use case-based [22, 23], and behavior-driven techniques [24, 25, 26, 27] mainly rely on source code or other artifacts, which might not always be available during the testing.

Regarding the aforementioned issues, we propose that machine learning techniques could tackle them. One category of machine learning algorithms is reinforcement learning (RL), which is mainly intended to train an agent (learner) how to solve a problem in a system through being rewarded or punished in a trial and error interaction with the system. Model-free RL is a subset of RL enabling the learner to explore the system behavior (the behavior of the software under test (SUT) in this case) and learn the optimal policy, to accomplish the objective (generating performance test cases resulting in the performance breaking point in this case) without access to source code and a model of the system. The learner can store the learned policy and is able to replay the learned policy in future situations, which can lead to efficiency improvement.

Goal of the paper. Our research goal is represented by the following question:

How can we adaptively and efficiently generate the performance test cases resulting in the performance breaking point for different software programs without access to the underlying source code and performance models?

Finding performance breaking point is a key purpose in robustness analysis, which is of great importance for many types of software systems, particularly in mission- and safety-critical domains [28]. Moreover, the question above is worth exploring also in applications specifically, such as resource management (scaling, provisioning and scheduling) for cloud services [29], performance prediction [30, 31], and performance analysis of software services in other areas [32, 33].

Contribution. In this paper, we present the design and experimental evaluation of a self-adaptive fuzzy reinforcement learning-based (SaFReL) performance testing framework focusing on efficiently and adaptively generating the (platform-based) test conditions leading to the performance breaking point for different software programs without access to source code and performance models. It is a smart testing framework since it is able to learn how to gen-

erate the intended test cases efficiently for software programs with different performance sensitivity to resources (i.e., CPU-, memory-, and disk-intensive programs) in order to find the performance breaking point. The learning mechanism is based on using Q-learning with fuzzy state modeling and an adaptive action selection strategy. Fuzzy state modeling is applied to deal with the issue of uncertainty in defining sharp boundaries in state modeling. The adaptive action selection strategy is intended to make the learning well-adapted to conditions. SaFReL assumes two phases of learning, i.e., initial and transfer learning. In the initial learning phase, it learns the optimal policy to generate the target performance test cases initially upon observing the behavior of the first SUT. Afterwards in the transfer learning, it reuses the learned policy for the SUTs with a performance sensitivity analogous to already observed ones while still keeping the learning running in the long term. We demonstrate that SaFReL works adaptively and efficiently on different sets of SUTs which are either homogeneous or heterogeneous in terms of their performance sensitivity.

Our experiments are based on simulating the performance behavior of 50 instances of 12 well-known programs as the SUTs. Those instances are characterized with various initial amounts of granted resources and different values of response time requirements. We use two evaluation criteria, i.e., efficiency and adaptivity, to evaluate our approach. We demonstrate the efficiency of the approach in generating the test cases which result in reaching the intended performance breaking point and also the behavioral sensitivity of the approach to the learning parameters. In particular, the self adaptive fuzzy RL-assisted testing approach meets the intended objective, i.e., reaches the intended performance breaking point, more efficiently compared to a typical stress testing technique which mainly generates the performance test cases based on changing the conditions, e.g., decreasing the availability of resources, by certain steps in an exploratory way. SaFReL leads to reduced cost (in terms of computation time) for performance test case generation by reusing the learned policy upon the SUTs with similar performance sensitivity. Moreover, it adapts its operational strategy to various SUTs with different performance sensitivity effectively while preserving the efficiency. To summarize, our contributions in this paper are:

- A smart performance testing framework (agent) that efficiently and adaptively generates the performance test cases leading to the intended performance breaking points for different software programs without access

to source code or system models. It uses fuzzy RL and an adaptive action selection strategy for generation of test cases, and implements two phases of learning:

  - Initial learning during which the agent learns the optimal policy for the first time,
  - Transfer learning during which the agent replays the learned policy in similar cases while keeping the learning running in the long term.

- A two-fold experimental evaluation, i.e., performance (efficiency and adaptivity) and sensitivity analysis of the approach. The evaluation is carried out based on simulating performance behavior of various SUTs. We also implement a performance prediction module along with our smart tester agent to conduct the experimentation.

Structure of the paper. The rest of the paper is organized as follows: Section 6.2 discusses the background concepts and motivations for the proposed self-adaptive learning-based approach. Section 6.3 presents an overview of the architecture of the proposed testing framework, while the technical details of the constituent parts are described in Sections 6.4 and 6.5. In Section 6.6, we explain the functions of the learning phases. Section 6.7 reports on the experimental evaluation involving the experiments setup, the details of the performance prediction module, and the experimentation's results. Section 6.8 discusses the results, the lessons learned during the experimentation and also the threats to the validity of the results. Section 6.9 provides a review on the related work, and finally Section 6.10 concludes the paper and discusses some future directions.

6.2 Motivation and Background

Performance analysis, realized through modeling or testing, is important for performance-critical software systems in various domains. Anomalies in performance behavior of a software system or violations of performance requirements are generally consequences of the emergence of performance bottlenecks at the system or platform levels [34, 35]. A performance bottleneck is a system or resource component limiting the performance of the system and

hinders the system from acting as required [36]. The behavior of a bottleneck component is due to some limitations associated with the component such as saturation and contention which make the component act as a bottleneck. A system or resource component saturation happens upon full utilization of its capacity or when the utilization exceeds a usage threshold [36]. Capacity expresses the maximum available processing power, service (giving) rate or storage size. Contention occurs when multiple processes contend for accessing a limited number of shared components including resource components like CPU cycles, memory and disk or software (application) components.

There are various application-, platform- and workload-based causes for emergence of performance bottlenecks [34]. Application-based causes represent issues such as defects in the source code or system architecture faults. Platform-based causes characterize the issues related to hardware resources, operating system and execution environment. High deviations from the expected workload intensity and similar issues such as workload burstiness are denoted by workload-based causes.

On the other hand, detecting violations of performance requirements and finding performance breaking points are challenging, particularly for complex software systems. To address these challenges, we need to find how to provide critical execution conditions which make the performance bottlenecks emerge. The focus of performance testing in our case, is to assess the robustness of the system and find the performance breaking point.

The effects of internal causes (i.e., application/architecture-based ones) could vary due to continuous changes and updates of the software, i.e., Continuous Integration/Continuous Delivery (CI/CD), and even vary for different platforms (execution environments) and under different workload conditions. Therefore, the complexity of SUT and a variety of affecting factors make it hard to build a precise performance model expressing the effects of all types of factors at play. This is a major barrier motivating the use of model-free learning-based approaches like model-free RL in which the optimal policy for accomplishing the objective could be learned indirectly through interaction with the environment (SUT). In our problem statement, we consider SUTs (with all involved affecting factors) as the varying environment, and the testing system learns the optimal policy to achieve the target, which is finding an intended performance breaking point, without access to a model of the environment. The testing system explores the behavior of the environment through

varying the platform-based (and workload-based in future work) test conditions, stores the learned policy and is able to later reuse the learned policy in similar situations, i.e., other SUTs with similar performance sensitivity to resource restriction. This is the interesting feature of the proposed learning approach which is supposed to lead to productivity benefits in terms of reduction of the testing system's effort, i.e., saving computation time.

Regarding the aforementioned challenges and strong points of the model-free learning-based approach, we hypothesize that in a CI/CD process based on agile software development, performance engineers and testers can save time and resources by using SaFReL for performance (stress) testing of various releases or variants. SaFReL provides an agile efficient performance test case generation technique (See Section 6.7 and Section 6.8 for efficiency evaluation) while eliminating the need for source code or system model analysis.

6.2.1 Reinforcement Learning

The model-free learning-based performance testing proposed in this paper is an interactive testing framework which involves an initial learning phase, and a subsequent transfer learning phase during which the learned policy can be reused for further SUTs with similar performance sensitivity. SaFReL is able to detect where it should use the previously learned policy and where it should change the strategy, update its policy and adapt to new situations, which means being self-adaptive to SUTs.

Reinforcement learning (RL) [37] is an interactive semi-supervised learning which has been frequently applied to building self-adaptive smart systems. It involves continuous interaction between the agent (learner) and the environment or the system which is controlled. The system/environment in our case is the SUT. The agent continuously senses the state of the system (at each time step). The state is modeled in terms of response time and resource utilization. Then, it takes an action, which is modifying the available resources capacity. After taking the action, it receives a reinforcement signal as a scalar reward which shows the effectiveness of the applied action to guide the agent towards the objective which is finding the intended performance breaking point. Figure 6.1 shows the interaction between the agent and the system which is the SUT in our case. The agent in an RL problem tries to find a policy which maximizes the total cumulative reward over the time. It uses an action selection strategy

based on a combination of trying out the available actions, i.e., exploration, and relying on the previously achieved experience to select the highly-valued actions, i.e., exploitation. For applying an RL-based approach to a problem, it is generally supposed that the environment is non-deterministic and also stationary upon transitions between the states of the system.

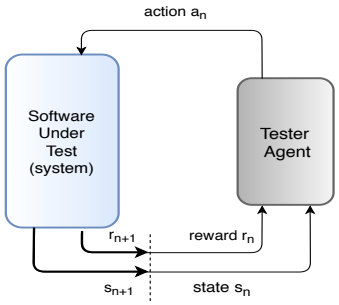


Figure 6.1: Interaction between agent and SUT in RL

Q-learning [37] is a well-known family of model-free RL algorithms in which a utility value of the long-term cumulative reward associated to the pairs of states and actions (state-actions) is learned. It is off-policy which means that the agent learns the optimal policy regardless of how the agent explores the environment. After learning the optimal policy, in the transfer learning phase, the agent replays the learned policy while keeping the learning running, i.e., occasionally exploring the action space and trying out different actions.

6.3 Architecture

This section provides an overview of the architecture of the proposed smart performance testing framework, SaFReL (see Figure 6.2). The entire interaction of the smart framework with each SUT, as a learning episode, consists of a number of learning trials. The steps of learning in each trial and the components involved in each step are described as follows:

1. Fuzzy State Detection. The fuzzification, fuzzy inference, and rule base components in Figure 6.2 are involved in state detection. The agent uses the

values of four quality metrics, i.e., 1) response time, and utilization improvements of 2) CPU, 3) memory, and 4) disk, to identify the state of the system, i.e., the SUT. In other words, the *state* expresses the status of the system relative to the target, i.e., the intended performance breaking point. In our case, these quality metrics are used to model (represent) the state space of the system. An ordinary approach for state modeling in RL problems is dividing the state space of the system into multiple mutually exclusive discrete sets. Each set represents a discrete state. At each time, the system must be at one distinct state. The relevant challenges of such crisp categorization, i.e., defining discrete states, include knowing how much a value is suitable to be a threshold for categories of a metric, and how we can treat the boundary values between categories. Instead of crisp discrete states, using fuzzy logic and defining fuzzy states can help address these challenges. We use fuzzy classification as a soft labeling technique for presenting the values of the metrics used for modeling the state of the system. Then, using a fuzzy inference engine and fuzzy rule base, the agent detects the fuzzy state of the system. More details about the fuzzy state detection of the agent are presented in Section 6.4.

2. Action Selection and Strategy Adaptation. After detecting the fuzzy state of the SUT, the agent takes an action. The actions are operations modifying the factors affecting the performance, i.e., the available resource capacity, in the current prototype. The agent selects the action according to an *action selection strategy* which it follows. The action selection strategy determines to what extent the agent should explore, i.e., to try out the available actions, and to what extent it should rely on the learned policy and select the action accordingly, i.e., to select a high-value action which has been tried and assessed before. The role of this strategy is guiding the action selection of the agent throughout the learning and is of importance for efficiency of the learning. In order to obtain desired efficiency, a proper trade-off between the exploration of the state action space and exploitation of the previously learned policy is critical. In our proposed framework, the smart agent is augmented by a *strategy adaptation* characteristic, as a meta-learning feature, which is responsible for dynamically adapting the degree of exploration and exploitation in various situations. This feature makes SaFReL able to detect where it should rely on the previously learned policy and where it should make a change in the strategy to update its policy and adapt to new situations. New situations mean acting

on new SUTs which are different from the previously observed ones in terms of performance sensitivity to resources. Software programs have different levels of sensitivity to resources. SUTs with different performance sensitivity to resources, e.g., CPU-intensive, memory-intensive or disk-intensive SUTs, will react to changes in resource availability differently. Therefore, when the agent observes a SUT which is different from the previously observed ones in terms of performance sensitivity, the strategy adaptation tries to guide the agent towards doing more exploration than exploitation. An indication of the resource sensitivity of SUT, i.e., being CPU-intensive, memory-intensive or disk-intensive, is an input to the strategy adaptation mechanism (see Figure 6.2). The components corresponding to action selection, stored experience which indicates the learned policy, and strategy adaptation are shown as yellow components in Figure 6.2. More details about the set of actions and the mechanism of strategy adaptation are described in Section 6.5.

3. Reward Computation. After taking the selected action, the agent receives a reward signal which indicates the effectiveness of the applied action to approach the target, i.e., finding the performance breaking point. The red block in Figure 6.2, i.e., reward computation component, calculates the reward (reinforcement) signal (see Section 6.5) for the applied actions.

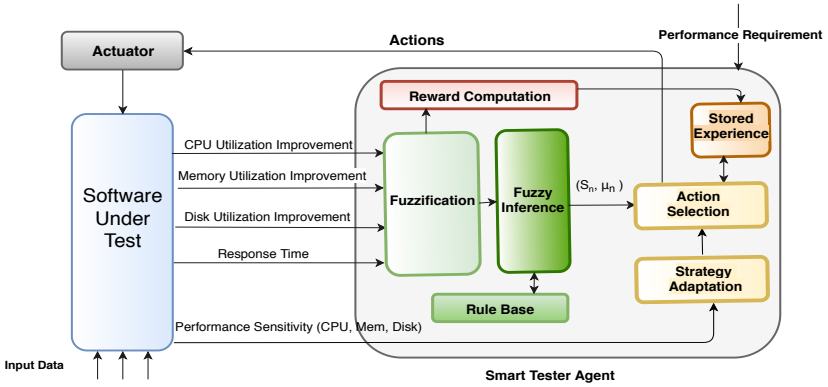


Figure 6.2: SaFReL architecture

6.4 Fuzzy State Detection

The state space of the system in our learning problem is modeled by quality measurements, i.e., CPU, memory and disk resource utilization improvement and response time of the SUT, as shown in Figure 6.3. Due to the uncertainty in defining sharp boundaries for the categories defined over the values of quality measurements, and to address this issue and preserve the desired precision of the model, fuzzy classification is applied to specify the categories. Therefore, the aggregate states of the system are fuzzy states which are inferred according to a fuzzy inference engine and a rule base [38, 39] (Figure 6.2). The system can be in one or more fuzzy states at the same time with different degrees of certainty. The details about the fuzzification of quality measurements, fuzzy inference to identify the state of the system, and the required steps and operations including fuzzy rules, fuzzy operators, and implication method, are described in Section 6.4.1.

6.4.1 Fuzzy State Space Modeling

As described in the previous section, a set of quality measurements, i.e., CPU, memory and disk utilization improvements and response time of the SUT, represent the state of the system. The values of these measurements are not bounded, then for simplifying the inference and also the exploration of the state space, we normalize the values of these parameters to the interval [0, 1] using the following functions:

$$RT_n = \frac{2}{\pi}\tan^{-1}(\frac{RT'_n}{RT^q}) \tag{6.1}$$

$$CUI_n = \frac{1}{CUI'_n} \qquad MUI_n = \frac{1}{MUI'_n} \qquad DUI_n = \frac{1}{DUI'_n} \tag{6.2}$$

where RT'_n, CUI'_n, MUI'_n, and DUI'_n are the measured values of the response time, CPU, memory and disk utilization improvements at time step n respectively and RT^q is the response time requirement. CUI'_n as the CPU utilization improvement is the ratio between the CPU utilization at time step n and its initial value (at the start of learning), that is, $CUI'_n = \frac{CU_n}{CU^i}$. Likewise, those are, $MUI'_n = \frac{MU_n}{MU^i}$ and $DUI'_n = \frac{DU_n}{DU^i}$. Using the normalization

function in Eq. 6.1, when $RT'_n = RT^q$ the normalized value of the response time, RT_n is 0.5, and for $RT'_n > RT^q$ the normalized values will be toward 1 and for $RT'_n < RT^q$ the normalized values will be toward 0. A tuple as $(CUI_n, MUI_n, DUI_n, RT_n)$ consisting of the normalized values of quality measurements is the input to the fuzzy state detection.

Fuzzification. Input fuzzification involves defining fuzzy sets and corresponding membership functions over the values of the quality measurements. A membership function is characterized by a linguistic term. A fuzzy set L is defined as $L = \{(x, \mu_L(x))|\ 0 < x, \quad x \in \mathbb{R}\}$ where a membership function $\mu_L(x)$ defines membership degrees of the values as $\mu_L : x \rightarrow [0, 1]$. Figure 6.3 shows the membership functions defined over the value domains of quality measurements. As shown in Figure 6.3, trapezoidal membership functions are used for *High* and *Low* fuzzy sets and a triangular counterpart for the *Normal* fuzzy set on the response time. In Figure 6.3, where RT^q is the requirement, a normal (medium) fuzzy set over the values of response time implies a small range around the requirement value as normal response time values. Moreover, in this case the ranges of membership functions were selected empirically and could be updated based on the requirements.

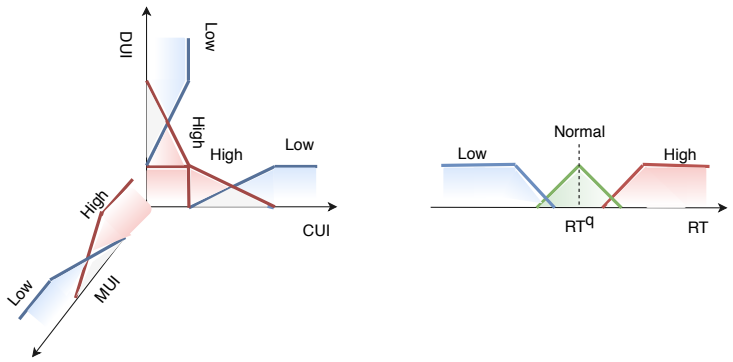


Figure 6.3: Fuzzy representation of quality measurements

Fuzzy Rules. After fuzzification of inputs, fuzzy rules based on domain knowledge, facilitate making inference at the next step, i.e., inferring the possible states that the system assumes. The fuzzy rules, as shown in Eq. 6.3, consist of two parts, i.e., antecedent and consequent. The former is a combination

of linguistic terms of the input parameters (normalized quality measurements) and the consequent is a fuzzy set with a membership function showing to what extent the system is in the associated state.

Rule 1: If CUI is High AND MUI is High AND DUI is Low AND RT is Normal, then State is HHLN. (6.3)

Fuzzy Operators. When the antecedents of the rules are made of multiple linguistic terms, which are associated to fuzzy sets, e.g., "High, High, Low and Normal", then fuzzy operators are applied to the antecedent to obtain one number showing the support or activation degree of the rule. Two well-known methods for the fuzzy AND operator are $minimum(min)$ and $product(prod)$. In our case, we use method min for the fuzzy AND operation. It shows that given a set of input parameters A, the degree of support for rule Ri is given as $\tau_{Ri} = \min_j \mu_L(a_j)$ where a_j is an input parameter in A and L is its associated fuzzy set in the rule Ri.

Implication Method. After obtaining the membership degree for the antecedent, the membership function of the consequent is reshaped using an implication method. There are also two well-known methods for implication process including $minimum(min)$ and $product(prod)$ which truncate and scale the membership function of the output fuzzy set respectively. The membership degree of the antecedent is given as input to the implication method. We use method min as implication method in our case. Based on the number of fuzzy sets specified over the values of the quality measurements and the possible combinations of them, which are associated to values of quality measurements, we define 24 rules in our rule base to define fuzzy states of the system (SUT). Then, the most effective rule, i.e., with the maximum support degree, is selected to determine the final fuzzy state of the system (S_n, μ_n). Figure 6.4 shows a representation of the fuzzy states of the system. Each of them represents one state for the system based on the fuzzy values (linguistic terms) assigned to quality measurements (CPU, memory and disk utilization improvement, and response time). Regarding the presentation of fuzzy states, L, H and N stand for Low, High and Normal terms respectively.

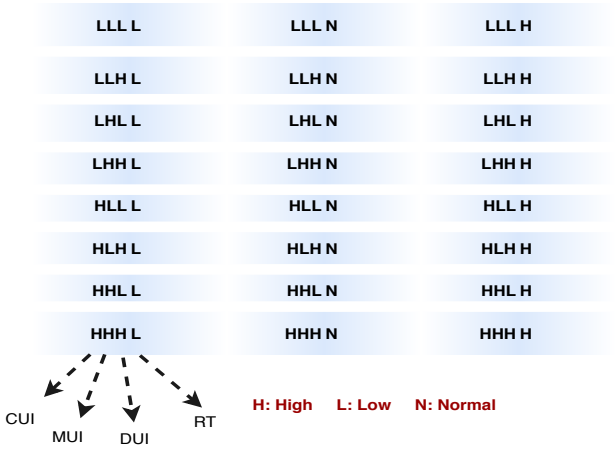


Figure 6.4: Fuzzy states of the system (SUT)

6.5 Adaptive Action Selection and Reward Computation

Actions. In SaFReL, the actions are the operations changing the platform-based factors affecting the performance, i.e., the available resources such as computation (CPU), memory and disk capacity. In the current prototype, the set of actions contains operations reducing the available resource capacity with finely tuned steps which are as follows:

$$\begin{aligned} AC_n = &\{\text{no action}\} \cup \{(CPU_n - y) \mid y \in CDF\} \\ &\cup \{(Mem_n - k) \mid k \in MDF_n\} \\ &\cup \{(Disk_n - k) \mid k \in MDF_n\} \end{aligned} \tag{6.4}$$

$$CDF = \{\frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1\} \tag{6.5}$$

$$MDF_n = \{(x \times \frac{Mem(Disk)_n}{4}) \mid x \in \{\frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1\}\} \tag{6.6}$$

where AC_n, CPU_n, Mem_n and $Disk_n$ represent the set of actions, the current available computation (CPU), memory and disk capacity at time step n respectively. The list of actions is as shown in Table 6.1.

Strategy Adaptation. The agent can use different strategies for selecting the actions. ε-greedy with different ε-values and Softmax are well-known methods for action selection in RL algorithms. They are intended to provide a right trade-off between exploration of the state action space and exploitation of the learned policy. In SaFReL, we use ε-greedy as the action selection strategy and the proposed strategy adaptation feature acts as a simple meta-learning algorithm which is intended to make changes to the ε value dynamically to make the action selection strategy well-adapted to new situations (new SUTs). Upon observing a SUT instance with a performance sensitivity different from the already observed ones, it adjusts the value of the parameter ε to direct the agent toward more exploration. On the other hand, upon interaction with SUT instances that are similar to the previous ones, the parameter ε is adjusted to increase exploitation. SaFReL detects the similarity between SUT instances by calculating *cosine similarity* between the performance sensitivity vectors of SUT instances, as shown in Eq. 6.7.

$$\begin{aligned} \text{similarity}(k, k-1) &= \frac{SV^k \; SV^{k-1}}{\|SV^k\| \|SV^{k-1}\|} \\ &= \frac{\sum_{i=1}^{3} SV_i^k SV_i^{k-1}}{\sqrt{\sum_{i=1}^{3} \left(SV_i^k\right)^2} \sqrt{\sum_{i=1}^{3} \left(SV_i^{k-1}\right)^2}} \end{aligned} \tag{6.7}$$

where SV^k represents the sensitivity vector of the k^{th} SUT instance and SV_i^k represents the i^{th} element of vector SV^k. The sensitivity vector contains the values of the resource sensitivity indicators of the SUT instance.

Table 6.1: Actions in SaFReL

Actions	
Operation	***Decrease***
Reducing memory / disk capacity	by a factor in MDF_n
Reducing computation (CPU) capacity	by a factor in CDF
No action	-

Reward Signal. The agent receives a reward signal indicating the effectiveness of the applied action in each learning step to guide the agent toward reaching the objective, which is finding the intended performance breaking point. We derive a utility function as a weighted linear combination of two functions indicating the response time deviation and resource usage, which is as follows:

$$U_n = \beta U_n^r + (1 - \beta) U_n^E \tag{6.8}$$

where U_n^r represents the deviation of response time from the response time requirement, U_n^E indicates the resource usage, and β, $0 \leq \beta \leq 1$ is a parameter intended to prioritize different aspects of stress conditions, i.e., response time deviation or limited resource availability. U_n^r is defined as follows:

$$U_n^r = \begin{cases} 0, & RT_n \leq RT^q \\ \frac{(RT_n - RT^q)}{(RT^b - RT^q)}, & RT_n > RT^q \end{cases} \tag{6.9}$$

where RT^q is the response time requirement and RT^b is the threshold defining the performance breaking point, i.e., the performance breaking point is reached when this threshold is exceeded. U_n^E represents the resource utilization in the reward signal, and is a weighted combination of the resource utilization values. It is defined using the following equation:

$$U_n^E = Sen^C CUI'_n + Sen^M MUI'_n + Sen^D DUI'_n \tag{6.10}$$

where CUI'_n, MUI'_n, and DUI'_n represent CPU, memory and disk utilization improvements respectively, and Sen^C, Sen^M and Sen^D are the performance sensitivity indicators of the SUT, and assume values in the range [0, 1].

6.6 Performance Testing using Self-Adaptive Fuzzy Reinforcement Learning

In this section, we describe details of the procedure of SaFReL to generate the performance test cases resulting in reaching the performance breaking points for various types of SUTs. The tester agent learns how to generate the target test cases for different types of software without access to source code or system models. The procedure of SaFReL, which includes initial and transfer learning phases, is as follows:

The agent measures the quality parameters and identifies the state- membership degree pair (S_n, μ_n) through the fuzzy state detection, where S_n is the fuzzy state of the system and μ_n indicates the membership degree, i.e., to what extent the system has assumed that state. Then, according to the action selection strategy, the agent selects one action, $a_n \in A_n$ based on the previously learned policy or through exploring the state action space. The agent takes the selected action and re-executes the SUT. In the next step, after the re-execution of the SUT, the agent detects the new state of the SUT, (S_{n+1}, μ_{n+1}) and receives a reward signal, $r_{n+1} \in \mathbb{R}$, indicating effectiveness of the applied action. After detecting the new state and receiving the reward, it updates the stored experience (learned policy). The whole procedure is repeated until meeting the stopping criterion, which is reaching the performance breaking point, (RT^b). The experience of the agent is defined in terms of the policy which the agent learns. A policy is a mapping between each state and action and specifies the probability of taking action a in a given state s. The purpose of the agent in the learning is to find a policy which maximizes the expected long-term reward achieved over the further learning trials, which is formulated as follows [37]:

$$R_n = r_{n+1} + \gamma r_{n+2} + ... + \gamma^k r_{n+k+1} = \sum_{k=0}^{\infty} \gamma^k r_{n+k+1} \tag{6.11}$$

where γ is a discount factor specifying to what extent the agent prioritize future rewards compared to the immediate one. We use Q-learning as a model-free RL algorithm in our framework. In Q-Learning, a utility value $Q^\pi(s, a)$ is assigned to each pair of state and action, which is defined as follows [37]:

$$Q^\pi(s, a) = E^\pi[R_n | s_n = s, a_n = a] \tag{6.12}$$

The q-values, $Q^\pi(s, a)$, form the experience base of the agent, on which the agent relies for the action selection. The q-values are updated incrementally during the learning. According to using fuzzy state modeling, we include the membership degree of the detected state of the system, μ_n^s, in the typical updating equation of q-values to take into account the impact of the uncertainty associated with the fuzzy state, which is as follows:

$$Q(s_n, a_n) = \mu_n^s[(1 - \alpha)Q(s_n, a_n) + \alpha(r_{n+1} + \gamma \max_{a'} Q(s_{n+1}, a'))] \tag{6.13}$$

where α, $0 \leq \alpha \leq 1$ is the learning rate which adjusts to what extent the new utility values affect (overwrite) the previous q-values. Finally, the agent finds

the optimal policy which suggests the action maximizing the utility value for a given state s:

$$a(s) = \underset{a'}{\operatorname{argmax}}\, Q(s, a') \tag{6.14}$$

The agent selects the action based on Eq. 6.14 when it is supposed to exploit the learned policy. SaFReL implements two learning phases, i.e., initial and transfer learning.

Initial learning. Initial learning occurs during the interaction with the first SUT instance. The initial convergence of the policy takes place upon the initial learning. The agent stores the learned policy (in terms of a Q-table, i.e., a table containing q-values). It repeats the learning episode multiple times on the first SUT instance to achieve the initial convergence of the policy.

Transfer learning. SaFReL goes through the transfer learning phase, after the initial convergence. During this phase, the agent uses the learned policy upon observing SUT instances with similar performance sensitivity to the previously observed ones, while keeping the learning running, i.e., updating the policy upon detecting new SUT instances with different performance sensitivity. Strategy adaptation is used in the transfer learning phase and makes the agent adapt to various SUT instances. Algorithms 5 and 6 present the procedure of SaFReL in both initial learning and transfer learning phases.

6.7 Evaluation

In this section, we present the experimental evaluation of the proposed self-adaptive fuzzy RL-based performance testing framework, SaFReL. We assess the performance of SaFReL, in terms of efficiency in generating the performance test cases resulting in the intended performance breaking points and adaptivity to various types of SUT programs, i.e., how well it can adapt its functionality to new cases while preserving its efficiency. We also evaluate the sensitivity of SaFReL to the learning parameters. The goal of the experimental evaluation is to answer the following research questions:

- RQ1. How efficiently can SaFReL generate the test cases leading to the performance breaking points for different software programs compared to a typical testing procedure?

Algorithm 5 SaFReL: Self-adaptive Fuzzy Reinforcement Learning-based Performance Testing

Required: S, A, α, γ;
Initialize q-values, $Q(s, a) = 0\ \forall s \in \mathbb{S},\ \forall a \in \mathbb{A}$ and $\varepsilon = \upsilon\ , 0 < \upsilon < 1$;
Observe the first SUT instance;
repeat
    Fuzzy Q-Learning (with initial action selection strategy, e.g., ε-greedy, initialized ε)
until *initial convergence*;
Store the learned policy;
Start the transfer learning phase;
while *true* **do**
    Observe a new SUT instance;
    Measure the similarity;
    Apply strategy adaptation, i.e., adjust the degree of exploration and exploitation (e.g., tuning parameter ε in ε-greedy);
    Fuzzy Q-Learning with adapted strategy (e.g., new value of ε);
end

- RQ2. How adaptively can SaFReL act on various software programs with different performance sensitivity?
- RQ3. How is the efficiency of SaFReL affected by changing the learning parameters?

The following sub-sections describe the proposed setup for conducting the experiments, the evaluation metrics, and the analysis scenarios designed for answering the above research questions.

6.7.1 Experiments Setup

In this study, we implement the proposed smart testing framework (agent) along with a performance simulation module which simulates the performance behavior of SUT programs under different execution conditions. The simulation module receives the resource sensitivity values and based on the amounts of resources demanded initially and the amounts of them which are granted

Algorithm 6 Fuzzy Q-Learning

repeat

1. Detect the fuzzy state-degree pair (S_n, μ_n) of the SUT;
2. Select an action using the action selection strategy (e.g., ε-greedy: select $a_n = \text{argmax}_{a \in \mathbb{A}} Q(s_n, a)$ with probability (1-ε) or a random a_k, $a_k \in \mathbb{A}$ with probability ε);
3. Take the selected action, re-execute the SUT;
4. Detect the new fuzzy state-degree (S_{n+1}, μ_{n+1}) of the system;
5. Receive the reward signal, R_{n+1};
6. Update the q-value of the pair of previous state and applied action $Q(s_n, a_n) = \mu_n^s[(1-\alpha)Q(s_n, a_n) + \alpha(r_{n+1} + \gamma \max_{a'} Q(s_{n+1}, a'))]$

until *meeting the stopping criteria (reaching performance breaking point)*;

after taking each action, estimates the program throughput using the following equation proposed by Taheri et al in [40]:

$$Thr_j = \frac{\frac{CPU_j^g}{CPU_j^i} Sen_j^C + \frac{Mem_j^g}{Mem_j^i} Sen_j^M + \frac{Disk_j^g}{Disk_j^i} Sen_j^D}{Sen_j^C + Sen_j^M + Sen_j^D} \times Thr_j^N \qquad (6.15)$$

where CPU_j^i, Mem_j^i and $Disk_j^i$ the amounts of CPU, memory and disk resources demanded by program j at the initial state and CPU_j^g, Mem_j^g and $Disk_j^g$ are the amounts of resources granted to program j after taking an action, which modifies the resource availability. Sen_j^C, Sen_j^M and Sen_j^D represent the CPU, memory and disk sensitivity values of program j, and Thr_j^N represents the nominal throughput of program j in an isolated, contention free environment. The response time of the program is calculated as $RT_j = \frac{1}{Thr_j}$ in the simulation module. Figure 6.5 presents the implementation structure including SaFReL along with the implemented performance simulation module. In our implementation, the performance simulation module simulates the performance behavior of the SUT program and the testing agent interacts with the simulation module to capture the quality measures which are used for state detection.

Table 6.2 shows the list of programs and the corresponding resource sensitivity values used in the experimentation, the table data obtained from [40]. The collection listed in Table 6.2 includes various CPU-intensive, memory-

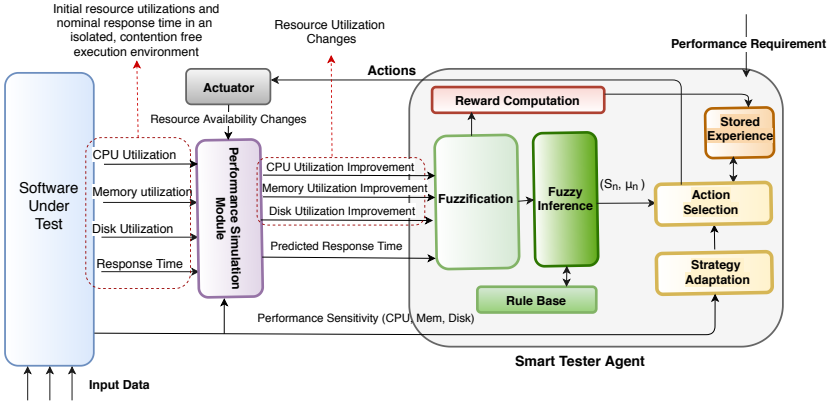


Figure 6.5: Implementation structure

intensive and disk-intensive types of programs and also the programs with combined types of resource sensitivity. The SUTs are instances of the programs listed in Table 6.2 and are characterized with various initial amounts of resources and also different values of response time requirements. Two analysis scenarios are designed to answer the evaluation research questions. The first one focuses on efficiency and adaptivity evaluation of the framework on various SUTs. In the second analysis scenario, the sensitivity of the approach to changes of the learning parameters are studied. The efficiency and adaptivity are measured (evaluated) according to following specification:

- *Efficiency* is measured in terms of number of learning trials required by the tester agent to achieve the target, i.e., reaching the intended performance breaking point. Number of learning trials is an indicator of the required computation time to generate the proper test case leading to the performance breaking point.

- *Adaptivity* is evaluated in terms of number of additional learning trials, i.e., computation time, which are required to re-adapt the learned policy to new observations for achieving the target.

Table 6.2: Programs and the corresponding sensitivity values used for experimental evaluation [40]

Programs	**Resource Sensitivity Values** (Sen^C, Sen^M **and** Sen^D)
Build-apache	(0.96, 0.04, 0.00)
n-queens	(0.97, 0.00, 0.00)
John-the-ripper	(0.96, 0.00, 0.00)
Apache	(0.97, 0.03, 0.00)
Dcraw	(0.48, 0.04, 0.00)
X264	(0.41, 0.02, 0.00)
Unpack-linux	(0.18, 0.09, 0.35)
Build-php	(0.97, 0.07, 0.00)
Blogbench	(0.11, 0.81, 0.18)
Bork	(0.00, 0.53, 0.20)
Compress-gzip	(0.00, 0.00, 0.47)
Aio-stress	(0.00, 0.30, 0.80)

6.7.2 Experiments and Results

Efficiency and Adaptivity Analysis

To answer RQ1 and RQ2, the performance of SaFReL is evaluated based on its efficiency in generating the performance test cases leading to the performance breaking points of different SUTs and its adaptation capability to new SUTs with performance sensitivity different from previously observed ones. Therefore, we examine the efficiency of SaFReL (in the transfer learning phase) compared to a typical common testing process for this target, which involves generating the performance test cases based on step-based changing of the availability of resources in an exploratory way, which is called *typical stress testing* hereafter. We select two sets of SUT instances: i) one including SUTs similar in the aspect of performance sensitivity to resources, i.e., similar wrt. the primarily demanded resource, (homogenous SUTs); and ii) the other set contains SUT instances different in performance sensitivity (heterogeneous SUTs). The SUT instances assume different initial amounts of CPU, memory and disk resources, and response time requirements. The amounts of resources, i.e., CPU,

memory and disk capacity, were initialized with different values in the range [1, 10] cores, [1, 50] GB, [100, 1000] GB respectively. The response time requirements range from 500 to 3000 ms. The intended performance breaking point for the SUT instances is defined as the point in which the response time exceeds 1.5 times the response time requirement.

In the efficiency analysis, we set the learning parameters, i.e., learning rate and discount factor to 0.1 and 0.5, respectively. We study the impacts of different variants of ε-greedy algorithm as the action selection strategy on the efficiency and adaptivity of the approach during the analysis. We investigate three variants of ε-greedy, i.e., $\varepsilon = 0.2$, $\varepsilon = 0.5$, and decaying ε, and also the proposed adaptive ε selection method.

Learning setup. First, we need to set up the initial learning. For choosing a proper a configuration for the action selection strategy in the initial learning, we evaluate the performance of different variants of ε-greedy algorithm, in terms of the number of required learning trials for initial convergence (Figure 6.6). For the initial convergence, we run the initial learning on the first SUT 100 times, i.e., 100 learning episodes. Table 6.3 presents a quick summarized view of the results, i.e., the average required learning trials regarding using different variants of ε-greedy. As shown in Figure 6.6 and Table 6.3, using ε-greedy with $\varepsilon = 0.2$ results in the fastest initial convergence, i.e., lowest average required trials.

After the initial convergence which occurs once, SaFReL is ready to act on various SUTs and is expected to be able to reuse the learned policy to meet the intended performance breaking points on further SUT instances, while still keeping the learning running. In fact, the optimal policy learned in the initial

Table 6.3: Initial convergence of SaFReL in the initial learning regarding using different variants of action selection strategy

	SaFReL - Initial Learning			
Action Selection Strategy: ε-greedy	$\varepsilon = 0.85$	$\varepsilon = 0.5$	$\varepsilon = 0.2$	decaying ε
Average number of learning trials (initial convergence)	22	22	11	16

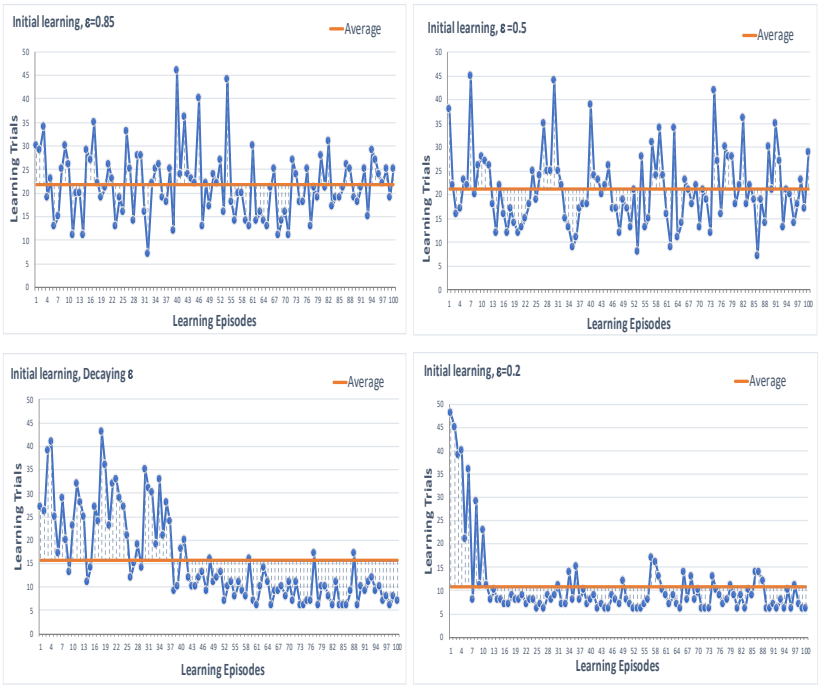


Figure 6.6: Initial convergence of SaFReL in 100 learning episodes during the initial learning

learning is not influenced by the used action selection strategy, since Q-learning is an off-policy algorithm [37], i.e., the learner finds the optimal policy independently of how the actions have been selected (action selection strategy). For the sake of efficiency, we choose the one that resulted in the fastest convergence.

In the following sections, first we investigate the efficiency of SaFReL compared to a typical stress testing procedure, when acting on homogeneous and heterogeneous sets of SUT instances, then its capability to adapt to new SUTs with different performance sensitivity while keeping its efficiency.

I. Homogeneous set of SUTs. We select CPU-intensive programs and make a homogeneous set of SUT instances during our analysis in this step. We

simulate the performance behavior of 50 instances of the CPU-intensive programs, i.e., Build-apache, n-queens, John-the-ripper, Apache, Dcraw, Build-php, X264, and vary both the initial amounts of resources granted and the response time requirements. Figure 6.7 shows the efficiency of SaFReL on a homogeneous set of CPU-intensive SUTs compared to a typical stress testing procedure regarding using ε-greedy with different values of ε. Table 6.4 presents the average number of trials/steps for generating the target performance test case in the proposed approach and the typical testing procedure. As shown in Figure 6.7, it keeps the number of required trials for $\approx 94\%$ of the SUTs below the average number of required steps in the typical stress testing. Table 6.5 shows the resulting improvement in the average number of required trials/steps for meeting the target, i.e., reduction in the required computation time, compared to the typical stress testing process.

In the transfer learning, the agent reuses the learned policy based on the allowed degree of policy reusing according to its action selection strategy in the transfer learning. As shown in Table 6.4, it implies that in the transfer learning the agent does fewer trials (based on the degree of allowed policy reusing) to meet the target on new cases, which leads to a higher efficiency. According to Table 6.5, on a homogeneous set of SUTs, more policy reusing leads to higher efficiency (more computation time improvement).

Table 6.4: Average number of trials/steps for generating the target performance test case on the homogeneous set of SUTs

	SaFReL with ε-greedy			
Approach	$\varepsilon = 0.5$	decaying ε	$\varepsilon = 0.2$	Typical stress testing
Average number of trials/steps	10	10	7	12

II. Heterogeneous set of SUTs. In this part of the analysis, to complete the answer to RQ1 and and also answer RQ2, we examine the efficiency and adaptivity of SaFReL during the transfer learning on a heterogeneous set of SUTs which includes various CPU-intensive, memory-intensive and disk-intensive

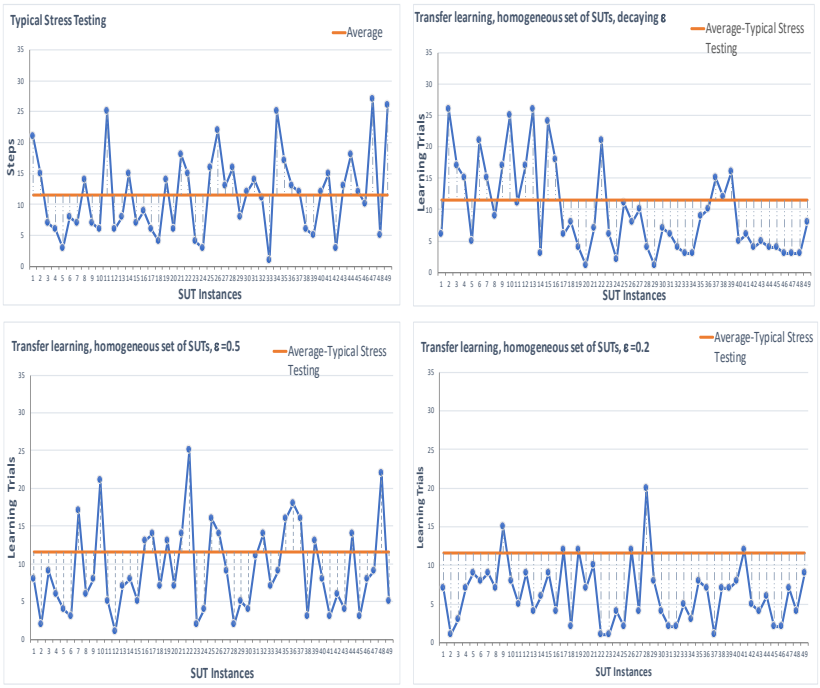


Figure 6.7: Efficiency of SaFReL on a homogeneous set of SUTs in the transfer learning

ones. We simulate the performance behavior of 50 SUT instances from the list of the programs in Table 6.2. We evaluate the efficiency of SaFReL on the heterogeneous set of SUTs compared to the typical stress testing procedure regarding using ε-greedy with $\varepsilon = 0.2, 0.5$, and decaying ε (Figure 6.8). As shown in Figure 6.8 the transfer learning algorithm with a typical configuration of the action selection strategy, such as $\varepsilon = 0.2, 0.5$ and decaying ε, which imposes a certain degree of policy reusing based on the value of ε does not work well. It does not outperform the typical stress testing, but also slightly degrades in some cases of ε. When the smart agent acts on a heterogeneous set of SUTs, blind replaying of the learned policy (i.e., just based on the value of ε) is not effective, and the tester agent needs to know where it should do policy reusing and where it requires more exploration to update the policy.

As described in Section 6.5, to solve this issue, i.e., to improve the perfor-

Table 6.5: Computation time improvement on the homogeneous set of SUTs

	SaFReL		
Action Selection Strategy: ε-greedy	$\varepsilon = 0.5$	decaying ε	$\varepsilon = 0.2$
Improvement in the number of trials	16%	16%	42%

mance of SaFReL when it acts on a heterogeneous set of SUTs, it is augmented with a simple meta-learning feature enabling it to detect the heterogeneity of the SUT instances and adjust the value of parameter ε, adaptively. In general, it implies that when the smart tester agent observes a SUT instance which is different from the previously observed ones wrt the performance sensitivity, it changes the action selection strategy to doing more exploration and upon detecting a SUT instance with the same performance sensitivity as the previous ones, it makes the action selection strategy strive for more exploitation. As illustrated in Section 6.5, the strategy adaptation module which fulfills this function, measures the similarity between SUTs at two levels of observations, then based on the measured values, adjusts the value of parameter ε. The threshold values of similarity measures and the adjustments for parameter ε in the experimental analysis are described in Algorithm 7.

Algorithm 7 Adaptive ε selection

```
if similarity_{k,k-1} ≥ 0.8 then
  if similarity_{k,k-2} ≥ 0.8 then
    ε ← 0.2
  else
    ε ← 0.5
  end if
else if similarity_{k,k-1} < 0.8 then
  ε ← 0.5
end if
```

Figure 6.9 shows the efficiency of SaFReL regarding the use of similarity detection and the adaptive ε-greedy action selection strategy on a heterogeneous set of SUTs. Regarding the use of adaptive ε selection, SaFReL makes a considerable improvement and is able to keep the number of required trials for

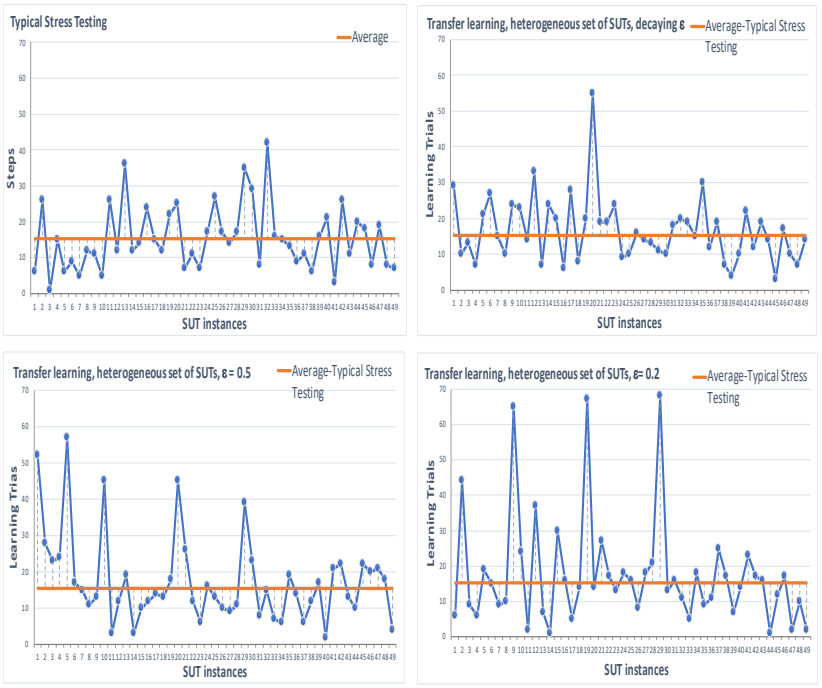


Figure 6.8: Efficiency of SaFReL on a heterogeneous set of SUTs regarding the use of typical configurations of ε-greedy

reaching the target on approximately $\approx 82\%$ of SUTs below the corresponding average value in the typical stress testing. Meanwhile, the average number of learning trials is totally lower than the typical stress testing procedure. Table 6.6 presents the average number of trials/steps for generating the target performance test case in SaFReL and the typical stress testing when they act on a heterogeneous set of SUTs. Table 6.7 shows the corresponding resulting improvement in the computation time respectively.

To answer RQ2, we investigate the adaptivity of SaFReL on the heterogeneous set of SUTs regarding the use of different variants of action selection strategy including adaptive ε selection (Figure 6.10). As shown in Figure 6.10, the number of required learning trials versus detected similarity is used to depict how adaptive SaFReL can act on a heterogeneous set of SUTs regarding

Table 6.6: Average number of trials/steps for generating the target performance test case on the heterogeneous set of SUTs

	SaFReL with ε-greedy				
Approach	$\varepsilon = 0.5$	decaying ε	$\varepsilon = 0.2$	adaptive ε	Typical stress testing
Average number of trials/steps	18	17	18	11	16

Table 6.7: Computation time improvement on the heterogeneous set of SUTs

	SaFReL			
Action Selection Strategy: ε-greedy	$\varepsilon = 0.5$	decaying ε	$\varepsilon = 0.2$	adaptive ε
Improvement in the number of trials	No	No	No	31%

the use of different configurations of ε. It shows that SaFReL with adaptive ε is able to adapt to changing situations, e.g., a mixed heterogeneous set of SUTs. In other words, on around $\approx 75\%$ of SUTs which are completely different from the previous ones (i.e., with $similarity_{k,k-1} < 0.8$) it still keeps the number of required trials to meet the target below the average value of the typical stress testing. It implies that it can act adaptively, i.e., reusing the policy wherever it is useful and doing more exploration wherever it is required.

Sensitivity Analysis

To answer RQ3, we study the impacts of the learning parameters including learning rate (α) and discount factor (γ), on the efficiency of SaFReL on both homogeneous and heterogeneous sets of SUTs. For conducting sensitivity analysis, we implement two sets of experiments that involve changing one learning parameter while keeping the other one constant. For the experiments running on a homogeneous set of SUTs, we use ε-greedy with $\varepsilon = 0.2$ as the well-suited variant of action selection strategy with respect to the results of ef-

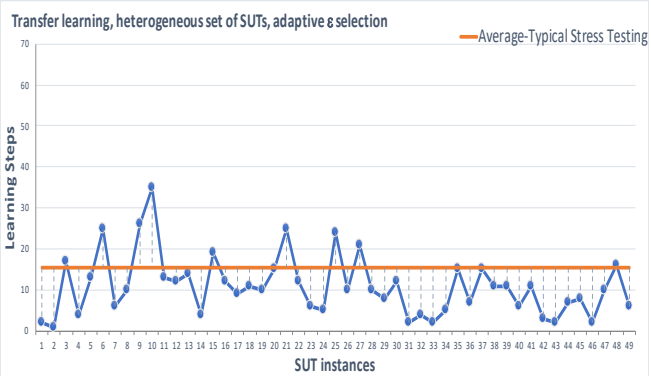


Figure 6.9: Efficiency of SaFReL on a heterogeneous set of SUTs regarding the use of adaptive ε-greedy action selection strategy

ficiency analysis (See Figure 6.7) and on the heterogeneous set of SUTs, we use adaptive ε selection (See Figure 6.9). During the sensitivity analysis experiments, to study the impact of the learning rate changes, we set the discount factor to 0.5. While examining the impact of the discount factor changes, we keep the learning rate fixed to 0.1. Figure 6.11 shows the sensitivity of SaFReL to changing learning rate and discount factor parameters when it acts on a homogeneous set of SUTs (CPU-intensive). Figure 6.12 depicts the results of the sensitivity analysis of SaFReL on a heterogeneous set of SUTs.

6.8 Discussion

Efficiency and Adaptivity Analysis. Using multiple experiments, we studied the efficiency of SaFReL compared to a typical stress testing procedure, on both a set of homogeneous and heterogeneous SUTs regarding the use of different action selection strategies.

The results of the experiments running on a set of 50 CPU-intensive SUT instances as a homogeneous set of SUTs, Figure 6.7 and Tables 6.4 and 6.5, show that using ε-greedy, $\varepsilon = 0.2$ as action selection strategy in the transfer learning leads to desired efficiency and an improvement in the computation time compared to the typical stress testing. It causes SaFReL to rely more on

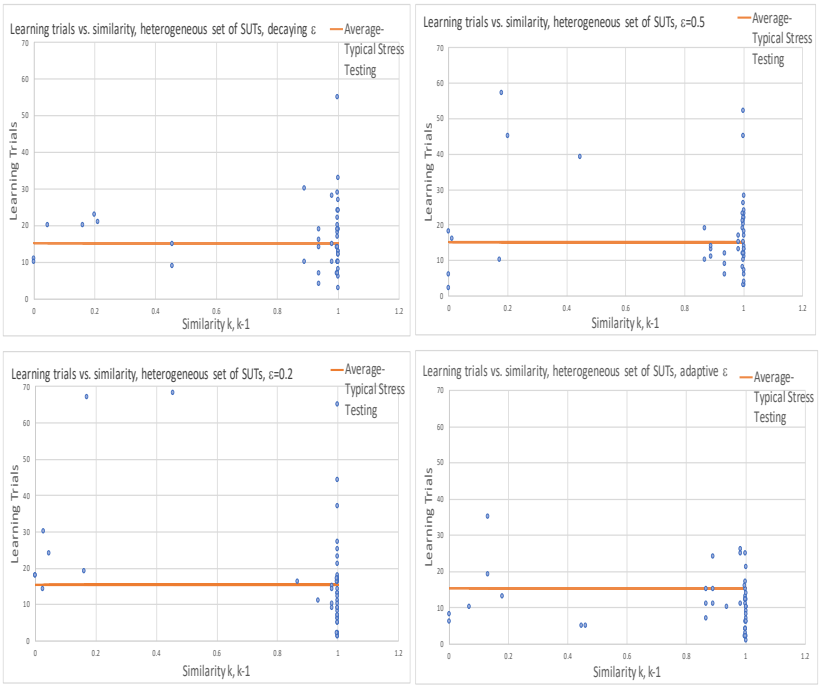
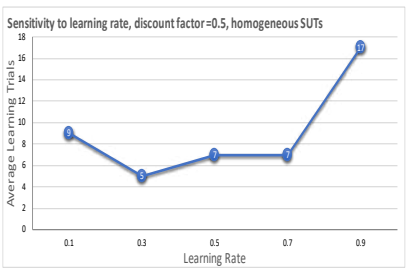
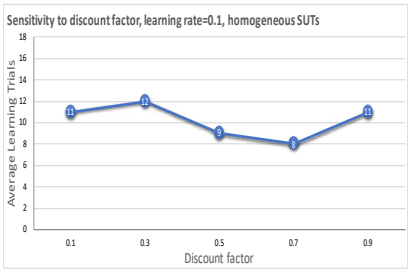


Figure 6.10: Adaptivity of SaFReL on a heterogeneous set of SUTs regarding the use of different variants of action selection strategy

reusing the learned policy and results in computation time saving. The existing similarity between the performance sensitivity of SUTs in a homogeneous set of SUTs makes the strategy of policy reusing successful in this type of testing situations.

Furthermore, we studied the efficiency of SaFReL on a heterogeneous set of 50 SUTs containing different CPU-intensive, memory-intensive and disk-intensive ones. The results of the analysis illustrate that choosing an action selection strategy without considering the heterogeneity among the SUTs does not lead to desirable efficiency. The typical variants of ε-greedy action selection strategy, do not work successfully compared to the typical stress testing. As shown in Figure 6.8 and Table 6.6 and 6.7, using a fixed strategy without considering the heterogeneity between SUTs could not perform efficiently.





Average Efficiency of SaFReL with $\varepsilon = 0.2$, Discount factor $\gamma = 0.5$					
	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
Average number of learning trials	9	5	7	7	17

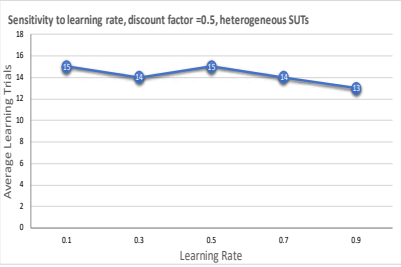
Average Efficiency of SaFReL with $\varepsilon = 0.2$, Learning rate $\alpha = 0.1$					
	$\gamma = 0.1$	$\gamma = 0.3$	$\gamma = 0.5$	$\gamma = 0.7$	$\gamma = 0.9$
Average number of learning trials	11	12	9	8	11

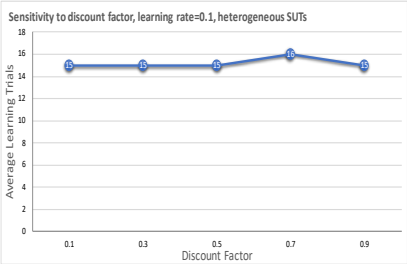
Figure 6.11: Sensitivity of SaFReL to learning rate and discount factor on the homogeneous set of SUTs

Then, we augmented our fuzzy RL-based approach with an adaptive action selection strategy which is a heterogeneity-aware strategy for adjusting the value of ε. It measures the similarity between the performance sensitivity of the SUTs and adjusts the ε parameter. As shown in Figure 6.9, using the adaptive ε-greedy, addressed the issue and led to an efficient generation of the target performance test case and a computation time improvement. It makes the agent able to reuse the learned policy according to the conditions, which means that it uses the learned policy wherever it is useful and does more exploration wherever it is required.

At the last part of the efficiency and adaptivity analysis, we extended our analysis by measuring the adaptivity of SaFReL when it performs on a heterogeneous set of SUTs. As shown in Figure 6.10, with the use of the adaptive ε-greedy, SaFReL is able to adapt to changing testing situations while preserving the efficiency.

Sensitivity Analysis. The results of the sensitivity analysis experiments on





Average Efficiency of SaFReL with adaptive ε, Discount factor $\gamma = 0.5$					
	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
Average number of learning trials	15	14	15	14	13

Average Efficiency of SaFReL with adaptive ε, Learning rate $\alpha = 0.1$					
	$\gamma = 0.1$	$\gamma = 0.3$	$\gamma = 0.5$	$\gamma = 0.7$	$\gamma = 0.9$
Average number of learning trials	15	15	15	16	15

Figure 6.12: Sensitivity of SaFReL to learning rate and discount factor on the heterogeneous set of SUTs

the homogeneous set of SUTs show that adjusting the learning rate to lower values such as 0.1 leads to better efficiency. Furthermore, regarding the sensitivity analysis of SaFReL to the discount factor on a homogeneous set of SUTs, the experimental results depict that lower values of the discount factor are suitable choices for the desired operation that we expect. However, the results of the sensitivity analysis on the heterogeneous set of SUTs do not show a considerable effects on the average efficiency of SaFReL when it acts on a heterogeneous set of SUTs regarding the use of adaptive ε-greedy.

Lessons Learned. The experimental evaluation of SaFReL depicts how machine learning can guide performance testing towards being automated and one step further, i.e., being autonomous. Common approaches for generating performance test cases mostly rely on source code or system models, but such development artifacts might not always be available. Moreover, drawing a precise model of a complex system which predicts the state of the system upon given performance-related conditions requires a solid endeavor. This

makes room for machine learning, particularly model-free learning techniques. Model-free RL is a machine learning technique enabling the learner to explore the environment (the behavior of the SUT in this case) and learn the optimal policy to accomplish the objective (finding the intended performance breaking point in this case) without having a model of the system. The learner stores the learned policy and is able to replay the learned policy in further suitable situations. This important characteristic of RL leads to a reduction in the effort of the learner to accomplish the objective in further cases and consequently leads to improved efficiency. Therefore, the main features that lead SaFRel to outperform an exploratory (search-based) technique are the capability of storing knowledge during the exploration and reusing the knowledge in suitable situations, and the possibility of selective and adaptive control on exploration and exploitation.

In general, automation, reduction of computation time and cost, and less dependency on source code and models are profound strengths of the proposed RL-assisted performance testing. Regarding applicability, according to the aforementioned strengths and the results of the experimental evaluation, the proposed approach could be beneficial to performance testing of software variants in software product lines, evolving software in continuous Integration/Delivery process and regression performance testing process.

Changes in Future Trends. With the emergence of serverless architecture, which incorporates third-party backend services (BaaS) and/or runs the server-side logic in state-less containers which are fully-managed by providers (FaaS), a slight shift in the objectives of performance evaluation, particularly performance testing on cloud-native applications is expected. Within the serverless architecture, the backend code is run without the need to manage and provision the resources on servers. For example in FaaS, scaling, including the resource provisioning and allocation, is automatically done by the provider whenever it is needed, to preserve the response time requirement of the application. In general, regarding the capabilities of new execution platforms and deployment architectures, the objectives of performance testing might be slightly influenced. Nevertheless, it is still crucial for a wide range of software systems.

Threats to Validity. Some of the main sources of threat to the validity of our experimental evaluation results are as follows:

Internal. There are a number of threats to internal validity of the results. The first source of threats is the formulation of the RL technique to address

the problem, which is very important for successful learning. Modeling the state space, actions and also the reward function are major players to guide the agent throughout the learning and make it learn the optimal policy. For example, boundaries defined in discrete states modeling are a threat to internal validity. To mitigate this threat, we used fuzzy labeling technique to deal with the issue of uncertainty in defining sharp values for boundaries. Regarding the actions, the formulation of actions affects the granularity of the exploration steps, thus we tried to define actions in a way to provide reasonable granularity for the exploration steps.

Another threat is the effect of the random selection in the action selection strategy. To alleviate this threat, we reported the average values of metrics during our experiments.

Finally, RL techniques like many other machine learning algorithms, are influenced by their hyperparameters such as learning rate and discount factor. During our efficiency and adaptivity analysis experiments, we did not change the learning parameters, we also conducted a set of controlled experiments to study the influence of learning parameters on the efficiency of our approach.

External. Model-free RL learns the optimal policy to achieve the target through interaction with the environment. Our approach was formulated based on an environment containing SUTs with three types of performance sensitivity, i.e., CPU-intensive, memory-intensive and disk-intensive, and our results are derived from the experimental evaluation of our approach on this environment. If the environment contains SUTs with other types of performance sensitivity such as network-intensive programs, then the approach needs to be reformulated slightly to support new types of performance sensitivities.

6.9 Related Work

Measurement of performance metrics under typical or stress test execution conditions, which involve both workload and platform configuration aspects [41, 42, 43, 44, 45], detection of performance-related issues such as functional problems or violations of performance requirements which emerge under certain workload or resource configuration conditions [46, 47, 48, 24] are common objectives of different types of performance testing.

Different approaches have been proposed to design the target performance test cases for accomplishing performance-related objectives such as finding in-

tended performance breaking points. Performance test conditions involve both workload and resource configuration status. A general high-level categorization of main techniques for generating the performance test cases is as follows:

Source code analysis. Deriving workload-based performance test conditions using data-flow analysis and symbolic execution are examples of techniques for designing fault-inducing performance test cases based on source code analysis to detect performance-related issues such as functional problems (like memory leaks) and performance requirement violations [49, 47].

System model analysis. Modeling the system behavior in terms of performance models like Petri nets and using constraint solving techniques [14], using the control flow graph of the system and applying search-based techniques [15, 16], and using other types of system models like UML models and using genetic algorithms [17, 18, 19, 20, 21] to generate the performance test cases are examples of the techniques based on system model analysis for generating performance test cases.

Behavior-driven declarative techniques. Using a Domain Specific Language (DSL) to provide declarative goal-oriented specifications of performance tests and model-driven execution frameworks for automated execution of the tests [25, 26, 27], and using a high-level behavior-driven language inspired from Behavior-Driven Development (BDD) techniques to define test conditions [24] in combination with a declarative performance testing framework like BenchFlow [26] are examples of behavior-driven techniques for performance testing.

Modeling the realistic conditions. Modeling the real user behavior through stochastic form-oriented models [22, 23], extracting workload characteristics from the recorded requests and modeling the user behavior using, e.g., extended finite state machine (EFSM) [50] or Markov chains [51], sandboxing services and deriving a regression model of the deployment environment based on the data resulting from sandboxing to estimate the service capacity [45], end-user clustering based on the business-level attributes extracted from usage data [52] are examples of techniques based on modeling the realistic conditions to generate the performance test cases.

Machine learning-enabled techniques. Machine learning techniques such as supervised and unsupervised algorithms mainly work based on building models and extracting patterns (knowledge) from the data. While, some other techniques such as RL algorithms are intended to train the learner agent to

solve the problems (tasks). The agent learns an optimal way to achieve an objective through interacting with the system. Machine learning has been widely used for analysis of data resulting from the performance testing and also for performance preservation. For example, anomaly detection through analysis of performance data, e.g., resource usage, using clustering techniques [53], predicting reliability from the testing data using Bayesian Networks [54], performance signature identification based on performance data analysis using supervised and unsupervised learning techniques [55, 56], and also adaptive RL-driven performance control for cloud services [57, 58, 59] and software on other execution platforms [60]. Machine learning has been also applied to the generation of performance test cases in some studies. For example, using symbolic execution in combination with an RL algorithm to find the worst-case execution path within a SUT [61], using RL to find a sequence of input workload leading to performance degradation [62], and a feedback-driven learning to identify the performance bottlenecks through extracting rules from execution traces [63]. There are also some adaptive techniques, which are slightly analogous to the concept of RL, for generating performance test cases. For example, an adaptive workload generation which adapts the workload dynamically based on some pre-defined adjustment policies [48], and a feedback-driven approach which uses search algorithms to benchmark an NFS server based on varying workload parameters to find the workload peak rate which satisfies the target response time confidence level.

6.10 Conclusion

Performance testing is a family of techniques commonly used as part of performance analysis, e.g., estimating performance metrics or detecting performance violations. One important goal of performance testing, particularly in mission-critical domains, is to verify the robustness of the SUT in terms of finding performance breaking point. Model-driven techniques might be used for this purpose in some cases, but drawing a precise model of the performance behavior of a complex software system under different application-, platform- and workload-based affecting factors is difficult. Furthermore, such modeling might disregard important implementation and deployment details. In software testing, source code analysis, system model analysis, use-case based design and behavior-driven techniques are some common approaches for generating

performance test cases. However, source code or other artifacts might not be available during the testing.

In this paper, we proposed a fuzzy reinforcement learning-based performance testing framework (SaFReL) that adaptively and efficiently generates the target performance test cases resulting in the intended performance breaking points for different software programs, without access to source code and system models. We used Q-learning augmented by fuzzy state modeling and an action selection strategy adaptation which resulted in a self-adaptive autonomous tester agent. The agent can learn the optimal policy to achieve the target (reaching the intended performance breaking point), reuse its learned policy when deployed to test similar software, and adapt its strategy when targeting software with different characteristics.

We evaluated the efficiency and adaptivity of SaFReL through a set of experiments based on simulating performance behavior of various SUT programs. During the experimental evaluation, we tried to answer how efficiently and adaptively SaFReL can perform testing of different SUT programs compared to a typical stress testing approach. We also performed a sensitivity analysis to explore how the efficiency of SaFReL is affected by changing the learning parameters.

We believe that the main strengths of using the intelligent automation offered by SaFReL are 1) efficient generation of test cases and reduction of computation time, and 2) less dependency on source code and models. Regarding applicability, we believe that SaFReL could be beneficial to the testing of software variants, evolving software during the (CI/CD) process and regression performance testing. Applying some heuristics and techniques to speed up the exploration of the state space like using multiple cooperating agents, and also extending the proposed approach to support workload-based performance test cases are further steps to continue this research.

Acknowledgment

This work has been supported by and received funding partially from the TESTOMAT, XIVT, IVVES and MegaM@Rt2 European projects.

Bibliography

[1] NS8. Did You Know A Slow Webpage Can Cost You 7% of Your Sales, 2018. Available at https://www.ns8.com/en/ns8u/did-you-know/a-slow-webpage-can-cost-you-7-percent-of-your-sales, Retrieved July, 2019.

[2] Elaine J Weyuker and Filippos I Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE transactions on software engineering*, 26(12):1147–1156, 2000.

[3] Andreas Brunnert, André van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Herbst, Pooyan Jamshidi, Reiner Jung, Joakim von Kistowski, et al. Performance-oriented devops: A research agenda. *arXiv preprint arXiv:1508.04752*, 2015.

[4] Leonid Grinshpan. *Solving enterprise applications performance puzzles: queuing models to the rescue*. John Wiley & Sons, 2012.

[5] ISO 25000. ISO/IEC 25010 - System and software quality models, 2019. Available at https://iso25000.com/index.php/en/iso-25000-standards/iso-25010, Retrieved July, 2019.

[6] Martin Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26. IEEE, 2007.

[7] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.

[8] Zhen Ming Jiang and Ahmed E Hassan. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, 41(11):1091–1118, 2015.

[9] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-based software performance analysis*. Springer Science & Business Media, 2011.

[10] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[11] Krishna Kant and MM Srinivasan. *Introduction to computer system performance evaluation*. McGraw-Hill College, 1992.

[12] Giovanni Denaro, Andrea Polini, and Wolfgang Emmerich. Early performance testing of distributed software applications. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 94–103. ACM, 2004.

[13] Pingyu Zhang, Sebastian Elbaum, and Matthew B Dwyer. Compositional load test generation for software pipelines. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 89–99. ACM, 2012.

[14] Jian Zhang and Shing Chi Cheung. Automated test case generation for the stress testing of multimedia systems. *Software: Practice and Experience*, 32(15):1411–1435, 2002.

[15] Yuanyan Gu and Yujia Ge. Search-based performance testing of applications with composite services. In *2009 International Conference on Web Information Systems and Mining*, pages 320–324. IEEE, 2009.

[16] Massimiliano Di Penta, Gerardo Canfora, Gianpiero Esposito, Valentina Mazza, and Marcello Bruno. Search-based testing of service level agreements. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1090–1097. ACM, 2007.

[17] Vahid Garousi. A genetic algorithm-based stress test requirements generator tool and its empirical evaluation. *IEEE Transactions on Software Engineering*, 36(6):778–797, 2010.

[18] Vahid Garousi. Empirical analysis of a genetic algorithm-based stress test technique. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1743–1750. ACM, 2008.

[19] Vahid Garousi, Lionel C Briand, and Yvan Labiche. Traffic-aware stress testing of distributed real-time systems based on uml models using genetic algorithms. *Journal of Systems and Software*, 81(2):161–185, 2008.

[20] Leandro T Costa, Ricardo M Czekster, Flávio Moreira de Oliveira, Elder de M Rodrigues, Maicon Bernardino da Silveira, and Avelino F Zorzo. Generating performance test scripts and scenarios based on abstract intermediate models. In *SEKE*, pages 112–117, 2012.

[21] Maicon Bernardino da Silveira, Elder de M Rodrigues, Avelino F Zorzo, Leandro T Costa, Hugo V Vieira, and Flávio Moreira de Oliveira. Generation of scripts for performance testing based on uml models. In *SEKE*, pages 258–263, 2011.

[22] Dirk Draheim, John Grundy, John Hosking, Christof Lutteroth, and Gerald Weber. Realistic load testing of web applications. In *Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 11–pp. IEEE, 2006.

[23] Christof Lutteroth and Gerald Weber. Modeling a realistic workload for performance testing. In *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 149–158. IEEE, 2008.

[24] Henning Schulz, Dušan Okanović, André van Hoorn, Vincenzo Ferme, and Cesare Pautasso. Behavior-driven load testing using contextual knowledge-approach and experiences. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pages 265–272. ACM, 2019.

[25] Vincenzo Ferme and Cesare Pautasso. A declarative approach for performance tests execution in continuous software development environments.

In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 261–272. ACM, 2018.

[26] Vincenzo Ferme and Cesare Pautasso. Towards holistic continuous software performance assessment. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, pages 159–164. ACM, 2017.

[27] Jürgen Walter, Andre van Hoorn, Heiko Koziolek, Dusan Okanovic, and Samuel Kounev. Asking what?, automating the how?: The vision of declarative performance engineering. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, pages 91–94. ACM, 2016.

[28] Kim Fowler. *Mission-critical and safety-critical systems handbook: Design and development for embedded applications*. Newnes, 2009.

[29] Brendan Jennings and Rolf Stadler. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 23(3):567–619, 2015.

[30] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: efficient performance prediction for large-scale advanced analytics. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 363–378, 2016.

[31] Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, Alexander Grebhahn, and Sven Apel. Tradeoffs in modeling performance of highly configurable software systems. *Software & Systems Modeling*, 18(3):2265–2283, 2019.

[32] Roberto Morabito. Virtualization on internet of things edge devices with container technologies: a performance evaluation. *IEEE Access*, 5:8835–8850, 2017.

[33] Zoran B Babovic, Jelica Protic, and Veljko Milutinovic. Web performance evaluation for internet of things applications. *IEEE Access*, 4:6974–6992, 2016.

[34] Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. Performance anomaly detection and bottleneck identification. *ACM Computing Surveys (CSUR)*, 48(1):4, 2015.

[35] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[36] Brendan Gregg. *Systems performance: enterprise and the cloud*. Pearson Education, 2013.

[37] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[38] Ludmila I Kuncheva. Fuzzy classifiers. *Scholarpedia*, 3(1):2925, 2008.

[39] MathWorks. Fuzzy Inference Process, 2019. Available at https://www.mathworks.com/help/fuzzy/fuzzy-inference-process.html, Retrieved July, 2019.

[40] Javid Taheri, Albert Y Zomaya, and Andreas Kassler. vmbbthrpred: A black-box throughput predictor for virtual machines in cloud environments. In *European Conference on Service-Oriented and Cloud Computing*, pages 18–33. Springer, 2016.

[41] Daniel A Menascé. Load testing, benchmarking, and application performance management for the web. In *Int. CMG Conference*, pages 271–282, 2002.

[42] James Hill, D Schmidt, James Edmondson, and Aniruddha Gokhale. Tools for continuously evaluating distributed system qualities. *IEEE software*, 27(4):65–71, 2009.

[43] Varsha Apte, TVS Viswanath, Devidas Gawali, Akhilesh Kommireddy, and Anshul Gupta. Autoperf: Automated load testing and resource usage profiling of multi-tier internet applications. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pages 115–126. ACM, 2017.

[44] Nicolas Michael, Nitin Ramannavar, Yixiao Shen, Sheetal Patil, and Jan-Lung Sung. Cloudperf: A performance test framework for distributed and dynamic multi-tenant environments. In *Proceedings of the*

8th ACM/SPEC on International Conference on Performance Engineering, pages 189–200. ACM, 2017.

[45] Anshul Jindal, Vladimir Podolskiy, and Michael Gerndt. Performance modeling for cloud microservice applications. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pages 25–32. ACM, 2019.

[46] Lionel C Briand, Yvan Labiche, and Marwa Shousha. Stress testing real-time systems with genetic algorithms. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1021–1028. ACM, 2005.

[47] Pingyu Zhang, Sebastian Elbaum, and Matthew B Dwyer. Automatic generation of load tests. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 43–52. IEEE Computer Society, 2011.

[48] Vanessa Ayala-Rivera, Maciej Kaczmarski, John Murphy, Amarendra Darisa, and A Omar Portillo-Dominguez. One size does not fit all: In-test workload adaptation for performance testing of enterprise applications. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 211–222. ACM, 2018.

[49] Cheer-Sun D Yang and Lori L Pollock. Towards a structural load testing tool. In *ACM SIGSOFT Software Engineering Notes*, volume 21, pages 201–208. ACM, 1996.

[50] Mahnaz Shams, Diwakar Krishnamurthy, and Behrouz Far. A model-based approach for testing the performance of web applications. In *Proceedings of the 3rd international workshop on Software quality assurance*, pages 54–61. ACM, 2006.

[51] Christian Vögele, André van Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. Wessbas: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems. *Software & Systems Modeling*, 17(2):443–477, 2018.

[52] Gururaj Maddodi, Slinger Jansen, and Rolf de Jong. Generating workload for erp applications through end-user organization categorization using high level business operation data. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 200–210. ACM, 2018.

[53] Mark D Syer, Bram Adams, and Ahmed E Hassan. Identifying performance deviations in thread pools. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 83–92. IEEE, 2011.

[54] Alberto Avritzer, Flávio P Duarte, Rosa Maria Meri Leao, Edmundo de Souza e Silva, Michal Cohen, and David Costello. Reliability estimation for large distributed software systems. In *Cascon*, page 12. Citeseer, 2008.

[55] Haroon Malik, Hadi Hemmati, and Ahmed E Hassan. Automatic detection of performance deviations in the load testing of large scale systems. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1012–1021. IEEE Press, 2013.

[56] Haroon Malik, Zhen Ming Jiang, Bram Adams, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. Automatic comparison of load tests to support the performance analysis of large enterprise systems. In *2010 14th European conference on software maintenance and reengineering*, pages 222–231. IEEE, 2010.

[57] Olumuyiwa Ibidunmoye, Mahshid Helali Moghadam, Ewnetu Bayuh Lakew, and Erik Elmroth. Adaptive service performance control using cooperative fuzzy reinforcement learning in virtualized environments. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 19–28. ACM, 2017.

[58] T Veni and S Mary Saira Bhanu. Auto-scale: automatic scaling of virtualised resources using neuro-fuzzy reinforcement learning approach. *International Journal of Big Data Intelligence*, 3(3):145–153, 2016.

[59] Pooyan Jamshidi, Amir Sharifloo, Claus Pahl, Hamid Arabnejad, Andreas Metzger, and Giovani Estrada. Fuzzy self-learning controllers for

elasticity management in dynamic cloud architectures. In *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*, pages 70–79. IEEE, 2016.

[60] Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. Adaptive runtime response time control in plc-based real-time systems using reinforcement learning. In *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 217–223. IEEE, 2018.

[61] Jinkyu Koo, Charitha Saumya, Milind Kulkarni, and Saurabh Bagchi. Pyse: Automatic worst-case test generation by reinforcement learning. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 136–147. IEEE, 2019.

[62] Tanwir Ahmad, Adnan Ashraf, Dragos Truscan, and Ivan Porres. Exploratory performance testing using reinforcement learning. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 156–163. IEEE, 2019.

[63] Mark Grechanik, Chen Fu, and Qing Xie. Automatically finding performance problems with feedback-directed learning software testing. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 156–166. IEEE, 2012.

Chapter 7

Paper C: Intelligent Load Testing: Self-Adaptive Reinforcement Learning-Driven Load Runner

Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Golrokh Hamidi, Markus Bohlin, Björn Lisper
Submitted to the 31st International Symposium on Software Reliability Engineering (ISSRE 2020), Coimbra, Portugal, March 2020.

Abstract

Load testing with the aim of generating an effective workload to identify performance issues is a time-consuming and complex challenge, particularly for evolving software systems. Current automated approaches mainly rely on analyzing system models and source code, or modeling of the real system usage. However, that information might not be available all the time or obtaining it might require considerable effort. On the other hand, if the optimal policy for generating the proper test workload resulting in meeting the objectives of the testing can be learned by the testing system, testing would be possible without access to system models or source code. We propose a self-adaptive reinforcement learning-driven load testing agent that learns the optimal policy for test workload generation. The agent can reuse the learned policy in subsequent testing activities such as meeting different testing targets. It generates an efficient test workload resulting in meeting the objective of the testing adaptively without access to system models or source code. Our experimental evaluation shows that the proposed self-adaptive intelligent load testing can reach the testing objective with lower cost in terms of the workload size, i.e., the number of generated users, compared to a typical load testing process, and results in productivity benefits in terms of higher efficiency.

7.1 Introduction

Performance is an important quality characteristic, which plays a key role in the success of software products, particularly in mission-critical domains where quality assurance of both functional and non-functional aspects of the behavior is of great importance. For example, enterprise applications (EAs) [1] with Internet-based user interfaces such as e-commerce websites are examples of systems whose success are subject to the performance assurance. EAs are often the core part of the corporate business organizations and their performance is a prerequisite for acceptable execution of business functions [2].

Performance, which is also called efficiency in some classifications of quality attributes [3, 4, 5], generally describes how well the system accomplishes its functionality. It presents time and resource bound aspects of a system's behavior which are often indicated by some common performance metrics (indicators) such as throughput, response time, and resource utilization. Performance assurance could be accomplished through conducting various analyses with different objectives, at different stages of the software development process. Performance analysis is conducted to meet the primary objectives, I) evaluating (measuring) performance metrics, II) detection of the functional problems emerging under specific execution conditions such as heavy workload, and III) detection of the violations of non-functional requirements [6]. The performance analysis is often performed under both typical and stress (extreme) execution conditions. The execution condition features different aspects of the execution environment like the resource availability and the workload under which the system operates.

Performance requirements violations often occur due to the emergence of performance bottlenecks [7, 8]. A performance bottleneck is generally defined as a system or resource component which limits the performance and hinders the system from performing as required [9]. Various application-, platform- and workload-based causes can lead to the emergence of performance bottlenecks [7].

Performance modeling and testing are the common approaches for conducting performance analysis. Modeling techniques using modeling notations such as queueing networks, Markov processes, and Petri nets, lead to a model of the system's behavior which is generally used to measure the performance metrics [10, 11, 12]. Performance testing as another family of techniques is

supposed to meet the objectives of performance analysis by executing the software under various realistic conditions.

Although models provide helpful insight of the performance of a system, there are still many details of the implementation and the execution environment that might be ignored in the modeling [13]. Moreover, drawing a precise model expressing the performance behavior under different and varying conditions is often impossible. Common testing approaches such as the techniques using source code analysis [14], system model, e.g., performance and UML model, analysis [15, 16, 17, 18, 19, 20] and also use case-based [21, 22] and behavior-driven [23, 24, 25, 26] performance testing approaches mostly rely on source code or system models. Nonetheless, those artifacts might not be available.

In this paper, we propose a self-adaptive model-free reinforcement learning load testing approach, which generates an efficient test workload resulting in the target performance breaking point without access to system model or source code. Performance breaking point refers to an execution condition under which the system becomes unresponsive or certain performance-related requirements are violated. In this study, the objective of testing is to meet a certain error rate threshold in received responses from a software under test (SUT). This work is a complementary system to our previously developed machine learning-assisted performance testing framework [27] called SaFReL. SaFReL [27] is a smart stress testing framework generating platform-based stress test conditions resulting in the target performance breaking points for different software programs without access to source code or system model; in the current paper, we address the generation of workload-based test conditions resulting in target performance breaking points.

We present the architecture and operational procedure of an intelligent reinforcement learning (RL)-assisted load testing agent which is able to learn the optimal policy to generate an efficient workload resulting in meeting an intended error rate threshold. Specifically, it uses Q-learning as the core learning algorithm with an adaptive action selection strategy to improve the learning performance. The proposed intelligent agent uses a load generator/runner tool like Apache JMeter to execute the workload on the SUT. We present a two-fold experimental evaluation, i.e., efficiency and sensitivity analysis, of the proposed approach on an e-commerce store as SUT. We demonstrate the efficiency of the proposed approach in generating the test workload which results

in the target error rate threshold and also the behavioral sensitivity of the approach to the learning parameters which influence the learning mechanism. In particular, the adaptive RL-assisted approach generates a more accurate and smaller workload (in terms of number of users) to meet the intended objective in comparison to a typical load testing technique which mainly involves applying a basic workload containing all the business transactions of the SUT, and increasing the load of the involved transactions (equally) upon a number of increase steps. In summary, generating an efficient test workload resulting in meeting the testing objective while reducing the dependency on system models and source code is the main achievement of using model-free RL to drive load testing in our proposed approach.

The rest of this paper is organized as follows: Section 7.2 discusses the motivation for applying model-free reinforcement learning to the problem and the primary concepts of RL. Section 7.3 presents the architecture and technical details of the proposed RL-assisted load testing approach. Section 7.4 presents the evaluation experiments and discusses the results. Section 7.5 gives an overview of the related work. The conclusion and future research directions are presented in Section 7.6.

7.2 Motivation and Background

Performance testing involves assessing the performance behavior of software/application under test (SUT) under stress or typical execution conditions. A typical load testing procedure involves running load generator tools (e.g., Apache Jmeter [28]) for a certain period of time and increasing the workload gradually or in some discrete steps. A typical workload is often configured as a set of concurrent (virtual) users doing some operations on the SUT which mimics the behavior of real users of the system. Any anomalies in the performance behavior of the system could be mainly a consequence of emerging bottlenecks at the level of platform or application [7, 8]. As mentioned, a bottleneck can be a system or resource component which makes the system fail or not perform as required [9]. It can happen due to the full utilization of the component capacity, exceeding a usage threshold or occurrence of contention [9].

Possible defects in source code or architecture and the issues related to platform resources and execution environment are often the root causes of emergence of bottlenecks. The application-based causes might vary during

the continuous integration/delivery (CI/CD) process and the platform-based ones might vary during the execution and also act differently under different workload conditions. Therefore, drawing a model which expresses the effects of all involved causes, is not easily possible. This issue is a motivation that makes room for model-free machine learning techniques such as model-free reinforcement learning (RL) [29] in which an intelligent agent can learn the optimal policy of achieving the intended objective without access to the model of the environment or the system which is under test. It is able to act to achieve the intended objective in an adaptive way to varying conditions. Furthermore, the learned policy can be reused in further stages of the testing. Based on these capabilities of model-free RL, in this paper, we aim at generating the workload-based test conditions resulting in reaching a target error rate threshold in responses of SUT without access to source code or system models.

7.2.1 Reinforcement Learning

Reinforcement learning is a learning technique that is mainly based on the interaction between the agent and the system (environment) of the problem. It has been widely used to build self-adaptive intelligent systems.

RL generally expresses how an agent learns the right behavior towards meeting the intended objective through being rewarded or punished in a trial and error interaction with the environment. At each step of the interaction, the agent observes the state of the system which is SUT in our case. It takes one possible action. The system undergoes changes upon receiving actions. The agent receives a reward signal showing how good the action was to accomplish the objective which was intended. The overall goal of the agent during the learning is modeled in terms of maximizing the cumulative long-term reward. In a model-free RL, the agent generally learns a policy to achieve this goal, i.e., maximizing the cumulative reward. It uses an action selection strategy to interact with and apply actions to the system. The action selection is often based on trying the available actions, i.e., exploration of the action space, or relying on the previously learned policy which leads to selecting highly-valued actions, i.e., exploitation of the action.

Q-learning [29] as a family of off-policy model-free RL algorithms focuses on learning a utility value of the pairs of states and actions. It is off-policy, which means that the learning takes place regardless of how the agent has cho-

sen the actions. The agent learns the optimal policy to achieve the target by learning the utility values and finding the optimal utility function. The learned policy is replayed by the agent while the agent generally keeps the learning running. It means that in addition to using experienced high-value actions, the agent sometimes explores the action space and tries out available different actions to keep the learned policy updated.

7.3 Reinforcement Learning-Assisted Load Testing

In this section, we present an overview of the architecture of our proposed RL-driven load testing solution and describe the technical details of the learning procedure. In summary, the proposed self-adaptive intelligent load testing features as follows:
How it addresses the problem. The proposed approach presents an adaptive smart load tester to accomplish the objectives of load testing, under different and varying execution conditions (e.g., different platform configurations) without access to system model or source code.
How it works. The proposed load tester learns the optimal policy to generate an efficient workload, in terms of a combination of workload sensitive and insensitive transactions, to reach the intended error rate threshold. The learned policy can be reused in further situations (stages) of testing, e.g., reaching different objective values of performance testing metrics, while the agent keeps the learning running in the long term.
How it learns. We use Q-learning, a model-free reinforcement learning algorithm [29], as our core learning technique. Fig. 7.1 shows the architecture of the proposed intelligent load testing solution. The main steps of each learning cycle of the RL-assisted load tester agent are detecting state, taking actions and computing reward (See Section **??**). We have formulated these steps of the RL algorithm as follows:

State Detection. Average response time and error rate, as two common performance metrics in load testing, are used to model the state of SUT. The agent measures these metrics at each learning trial and identifies the state of SUT. The values of these metrics are classified into some categories and labeled

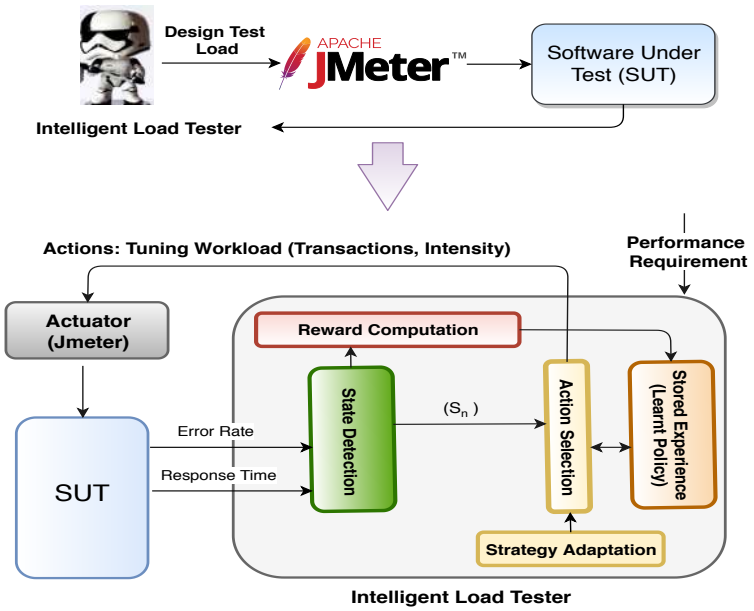


Figure 7.1: Architecture of RL-driven load testing solution

with the discrete classes as *Low*, *Normal* and *High* for response time and *Low* and *High* for error rate, then the combinations of those classes form the discrete states of the system, as shown in Fig. 7.2.

Actions. At each learning trial, the tester agent takes one of the possible actions after detecting the state of SUT. We define actions as adjusting the workload in terms of changing the load of the involved transactions, i.e., the numbers of users running the transactions. The list of the involved transactions is application-specific. Running each transaction, for example a transaction on a web application, involves sending a batch of requests from the client to the server. Many of the load generator tools such as Apache JMeter have the functionality of recording the involved requests in running a transaction of a SUT, to make an executable test plan for testing the SUT.
In our case, before the start of the learning, the set of involved requests in running each transaction of the SUT is extracted through recording an ordinary usage of the SUT by a user. Each transaction might also have some functional

dependency, i.e., some certain transactions are required to be done before that. Therefore, the whole set of involved requests including prerequisites in doing each transaction are extracted separately. Table 7.1 lists some of the common transactions for an online shop application which is used as SUT in our study. For example, the functional prerequisites for transaction *Add to cart*, are doing login, accessing the search page, and selecting the product. Therefore, each transaction of the workload is considered together with its functional dependencies.

Then, the set of actions is defined as follows:

$$ActionList = \{\cup\, action_k,\ 1 \leq k \leq |\textit{List of Transactions}|\} \tag{7.1}$$

$$\begin{aligned} action_k = \{\cup\,(W_n^{T_j}) \mid\ & W_n^{T_j} = W_{n-1}^{T_j},\ \ if\ j \neq\ k, \\ & W_n^{T_j} = W_{n-1}^{T_j}\ +\ \frac{W_{n-1}^{T_j}}{3},\ if\ j = k, \\ & T_j \in \textit{List of transactions}, \\ & 1 \leq j \leq |\textit{List of Transactions}|\} \end{aligned} \tag{7.2}$$

where T_j indicates a transaction of the SUT. $W_n^{T_j}$ indicates the load of the transaction T_j at time step (learning cycle) n, i.e., the number of users running this transaction. Running this transaction includes running the operations which are prerequisite to the transaction. For example, running transaction $T_{addtocart}$, includes the operations *Login page*, *Login*, *Search page*, *Select product* and, *Add to cart*. The modified workload in each action runs for a certain period which can be defined empirically.

Reward Signal. After taking the selected action and running the modified workload, the tester agent receives a reward signal which shows how effective the applied action was to lead to meeting the target, i.e., reaching the intended error rate threshold. We define a function to represent the reward signal as follows:

$$r_n = \frac{(R_n^{av}\ -\ R_{n-1}^{av})}{R_n^{av}}\ +\ \frac{(er_n^{max}\ -\ er_{n-1}^{max})}{er_n^{max}} \tag{7.3}$$

where R_n^{av} and R_{n-1}^{av} indicate the average response time of the SUT at step n and step $n-1$ of the learning, while er_n^{max} and er_{n-1}^{max} represent the maximum error rate hit at step n and step $n-1$ of the learning respectively.

Table 7.1: Common transactions in an online shop

Operation	***Description***
Home	Access to home page
Sign up page	Access to Sign up page
Sign up	Register and add a new user
Login page	Access to login page
Login	Sign in at the system
Search page	Access to search page
Select product	See the details of the selected product
Add to cart	Add the selected product to the cart
Payment	Access to payment page
Confirm	Confirm the order (payment)
Log out	Log out

Learning Procedure. In RL, the agent is intended to learn the optimal policy to accomplish the objective of the problem. Policy decides which action to be taken by the agent, given a certain state. In model-free RL, there are generally two approaches to realize this: learning the policy directly and indirectly. In Q-learning algorithm, the agent learns an optimal utility function, i.e., action-value function $Q^*(s, a)$, from which the optimal policy can be obtained. The optimal action-value function, $Q^*(s, a)$, gives the expected long-term reward (return), given state s, taking an arbitrary action a and then following the optimal policy, which is as follows:

$$Q^*(s, a) = \underset{\pi}{\operatorname{argmax}}\, E^{\pi}[R_n | s_n = s, a_n = a] \tag{7.4}$$

$$R_n = \sum_{k=0}^{\infty} \gamma^k r_{n+k+1} \tag{7.5}$$

where γ is a discount factor for future rewards and R_n is the discounted long-term return in terms of cumulative reward. In general, the optimal policy selects

the action maximizing the expected return given starting from state s. Moreover, according to the definition of $Q^*(s,a)$, given Q^*, the optimal action for state s, $a^*(s)$ is obtained as:

$$a^*(s) = \underset{a'}{\operatorname{argmax}}\, Q^*(s, a') \tag{7.6}$$

In order to obtain optimal policy, Q-values are stored (Q-table) and considered as the experience of the agent. During the learning, the Q-values are updated incrementally according to Eq. 7.7:

$$Q(s_n, a_n) = (1-\alpha)Q(s_n, a_n) + \alpha[r_{n+1} + \gamma \max_{a'} Q(s_{n+1}, a') \tag{7.7}$$

where α, $0 \leq \alpha \leq 1$ adjusts the rate of learning which controls the impact of new Q-values on the previous ones.

Learning phase. The research problem of this study, i.e., generating an efficient workload to reach an intended load testing objective, can be considered as a sequential decision-making problem and RL learning could be a beneficial solution to this problem, since the SUT (environment) and execution platform is supposed to be initially unknown to the tester agent. Then, in RL learning, the problem is formulated in terms of a Markov Decision Process (MDP) and the agent finds (learns) the optimal policy to generate an efficient workload to reach the intended error rate through solving the MDP. One of the basic paradigms in solving an MPD is finding an optimal policy through policy and value iterations techniques. The value iteration technique has been used in this study. Algorithms 8 and 9 present the procedure of the proposed RL-driven intelligent load testing.

Applicability. The proposed solution generates a proper workload to meet the intended objective more efficiently than a typical load testing procedure. It means that it generates a more accurate, fine-grained and smaller target workload (in terms of the number of users) in comparison to a typical load testing process, which results in saving more time and cost in the testing. This achievement is beneficial to various testing situations such as stress testing where a heavy workload is often applied to meet the objectives and regression load testing where testing scenarios are required to be repeated after applying any changes. The pay-as-you-go cost of many of the commercial load generation tools is based on the number of generated virtual users. Therefore, The

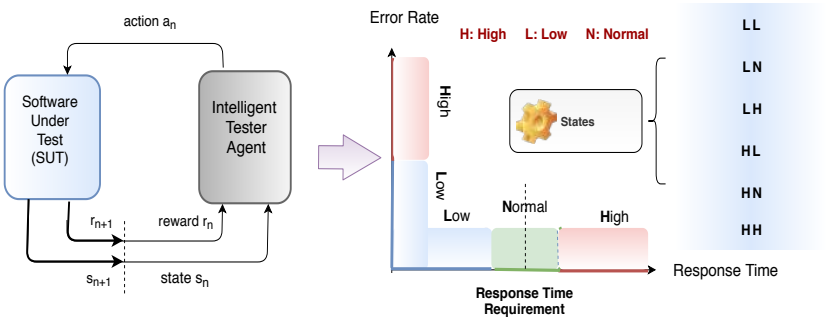


Figure 7.2: States of SUT in the proposed RL-based load testing

efficient generated workload by the proposed RL-driven approach could lead to considerable cost and time-saving in the testing process. Moreover, the proposed intelligent agent is able to reuse the learned policy in further similar testing scenarios, for example, reaching different values of error rate, i.e., transfer learning. It reuses the learned policy during the transfer learning, while it also keeps the learning running to keep the learned policy updated upon changes in the environment. Reusing the learned policy could also result in cost saving in terms of reducing the required effort for generating the proper workload in further situations.

7.4 Results and Discussion

This section discusses the results of the experimental evaluation of our proposed intelligent RL-assisted load testing. The purposes of the evaluation experiments are to assess the proposed approach in terms of how efficient it performs in meeting the intended objective of the testing and how sensitive its behavioral performance is to the learning parameters. We use the size of the generated workload as a key performance indicator (KPI) for evaluating the efficiency of the approach in comparison to a typical load testing procedure. We show how the learning-assisted approach can perform efficiently in an adaptive way in different situations without access to the system model or source code of the SUT.

Algorithm 8 Adaptive Reinforcement Learning-Driven load Testing

Required: S, A, α, γ;
Initialize q-values, $Q(s,a) = 0\ \forall s \in \mathbb{S},\ \forall a \in \mathbb{A}$ and $\varepsilon = \upsilon\ , 0 < \upsilon < 1$;
while *Not (initial convergence reached)* **do**
    Q-Learning (with initial action selection strategy, e.g., ε-greedy, initialized ε)
end
Store the learned policy;
Adapt the action selection strategy to transfer learning, i.e., tune parameter ε in ε-greedy;
while *true* **do**
    Q-Learning with adapted strategy (e.g., new value of ε);
end

Algorithm 9 Q-Learning

repeat
    1. Detect the state (S_n) of the SUT;
    2. Select an action (See Eq. 7.1) according to the action selection strategy, e.g., ε-greedy: select $a_n = \text{argmax}_{a \in \mathbb{A}}\, Q(s_n, a)$ with probability (1-ε) or a random a_k, $a_k \in \mathbb{A}$ with probability ε;
    3. Take the selected action: Tune the workload and run the modified workload on the SUT;
    4. Detect the new state (S_{n+1}) of the SUT;
    5. Compute the reward, R_{n+1};
    6. Update the Q-value of the pair of previous state and taken action $Q(s_n, a_n) = (1-\alpha)Q(s_n, a_n) + \alpha[r_{n+1} + \gamma \max_{a'} Q(s_{n+1}, a^{'})]$
until *meeting the stopping criteria (reaching the intended error rate)*;

7.4.1 Evaluation Setup

In this study, we implement our approach based on using ε-greedy as an action selection strategy with four configurations as $\varepsilon = 0.2, 0.5, 0.85$ and decaying ε in which the value of ε decreases gradually during the learning. ε-greedy is one of the well-known methods for action selection in RL algorithms. The action

selection strategy is intended to provide a right trade-off between exploration of the state-action space and exploitation of the learned policy. In ε-greedy, the value of ε adjusts the degree of exploration versus exploitation, as it leads the agent to select a high-value action based on its learned policy with probability (1-ε) or a random possible action with probability ε, given a certain state. We use Apache JMeter 5.1.1 as an actuator to execute the test workload. The intelligent tester agent designs the test workloads and Apache JMeter executes them. In the proposed learning approach, we set the response time requirement and error rate threshold for classifying the states of the SUT to $500ms$ and 0.5 respectively, which could be also determined empirically based on the SUT requirements.

In the experimental evaluation, as SUT, we use an e-Commerce store (online shop) created by WordPress themes and deployed on a shared WordPress hosting server with one shared CPU, up to 512MB shared RAM and 100GB storage. The SUT supports 11 common business use cases that have been described in Table 7.1. The objective of the testing scenario in our experiments is reaching an intended level of *error rate*, which is 40%, in our analysis scenarios.

We evaluate the proposed approach through two analysis scenarios, i.e efficiency, and sensitivity analysis. In the efficiency analysis, we study the efficiency of the proposed approach in generating a target test workload meeting the intended objective. We compare the efficiency of the approach with a typical load testing approach. A typical load testing procedure for accomplishing the aforementioned objective involves applying a basic workload that contains all the transactions with the *same* number of users per each, then applying an increase of 33% to the load of each transaction equally upon a number of steps. The performance of the proposed approach in terms of size of the generated workload, i.e., the number of users, during multiple executions of the learning algorithm (learning episodes) is demonstrated. During the sensitivity analysis, we study the performance sensitivity of the approach to the learning parameters such as *learning rate* and *discount factor* through systematically controlled experiments which involve changing one parameter while keeping the other ones constant.

7.4.2 Experiments and Results

Efficiency Analysis. In the efficiency analysis, we set the learning rate and discount factor to 0.1 and 0.5 respectively. Moreover, the performance-related status of the hosting server, i.e., the resource availability, remains unknown to the intelligent load tester. Each experiment starts with running a baseline workload and trials indicate the number of steps that the workload was modified (taken actions) by the intelligent agent. We run each experiment 10 times.

Figure 7.3 shows the efficiency of the self-adaptive RL-assisted load testing in terms of size of the generated workload meeting the intended error rate threshold compared to a typical load testing procedure regarding using ε-greedy with different values of ε. The adaptive RL-assisted learns the optimal policy to generate an efficient test workload meeting the objective. It stores the experienced trajectory during the learning and is also able to exploit it in further situations. Despite all the varying conditions on the hosting, it tries to adapt test workload well to meet the intended objective. As shown in figure 7.3, the proposed approach is able to meet the error rate threshold with less number of generated users than typical load testing, which means lower cost and time in the testing. Table 7.2 presents the average size of the generated workload and the number of required learning trials or steps for generating the target workload.

The performance of the adaptive RL-assisted approach regarding using different settings in ε-greedy action selection strategy is described as follows:
ε-greedy with $\varepsilon = 0.2$. It makes the intelligent agent mainly rely on the stored experience rather than exploring new actions. It might slow down the learning convergence and consequently reduce the adaptivity in a highly varying environment which more exploration is needed. This issue is also observable in terms of some high spikes in figure 7.3.
ε-greedy with $\varepsilon = 0.5$. This configuration provides a balance between the exploitation of the learned policy and the exploration of new actions. It gives a better efficiency than $\varepsilon = 0.2$ strategy in terms of size of the generated workload.
ε-greedy with $\varepsilon = 0.85$. It guides the intelligent agent towards doing more exploration which makes the performance of the approach closer to a search-based technique. It leads to a fairly good efficiency especially for highly changeable environments.

Table 7.2: Average size of the generated workload and the number of required trials

	RL-assisted with ε-greedy				
Approach	$\varepsilon = 0.85$	$\varepsilon = 0.5$	$\varepsilon = 0.2$	decaying ε	Typical load testing
Average workload size	52	50	80	46	135
Average number of trials	16	26	16	9	5

ε-greedy with decaying ε. This setting decreases ε gradually during the learning time. It makes the agent explore new actions more during the early stages of the learning and do more exploitation of the learned policy in the later steps of the learning. It provides the most promising efficiency in terms of both the size of the generated workload and the required trials for generating the target workload.

Figure 7.4 shows the output of the learning trials in a learning episode (for example using ε-greedy with $\varepsilon = 0.85$) and presents how the RL-assisted approach finds an efficient workload reaching the intended error rate in a number of learning trials.

Sensitivity Analysis. In this analysis, we study the behavioral sensitivity of the proposed approach to the learning parameters, i.e., learning rate (α) and discount factor (γ). We select ε-greedy with decaying ε as the action selection strategy, as it led to the most promising result in the efficiency analysis scenarios. We examine the effects of learning rate (α) and discount factor (γ) on the behavioral performance of the approach in a systematic way. Figure 7.5 shows the behavioral performance of the proposed approach regarding changing the values of learning rate and discount factor parameters. In the sensitivity analysis, we consider $\alpha = 0.1$ and $\gamma = 0.5$ as the baseline values and in four controlled experiments, we study the effects of changing the values of these parameters. First, we set the learning rate to 0.5 and 0.9 respectively while keeping the value of discount factor to 0.5, then in the second set of experiments we set the discount factor to 0.1 and 0.9 respectively, while fixing the learning rate at 0.1. As shown in figure 7.5, setting the learning rate to a

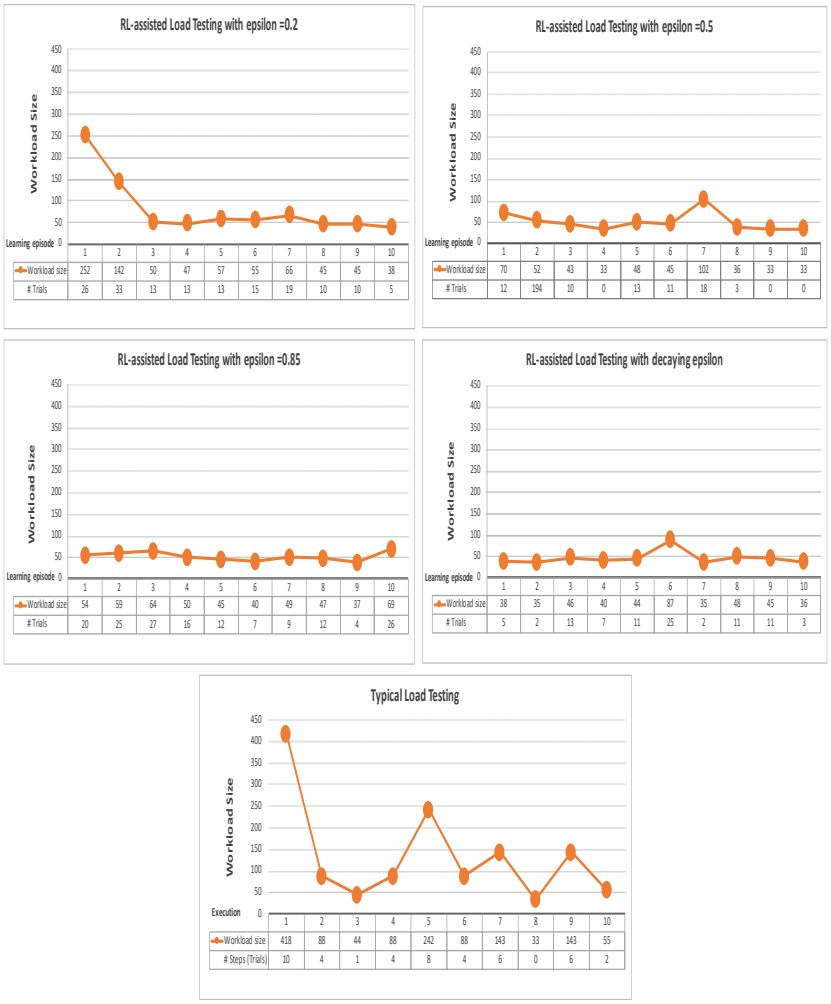


Figure 7.3: Size of the generated workload in the executions of learning-assisted and typical load testing approaches

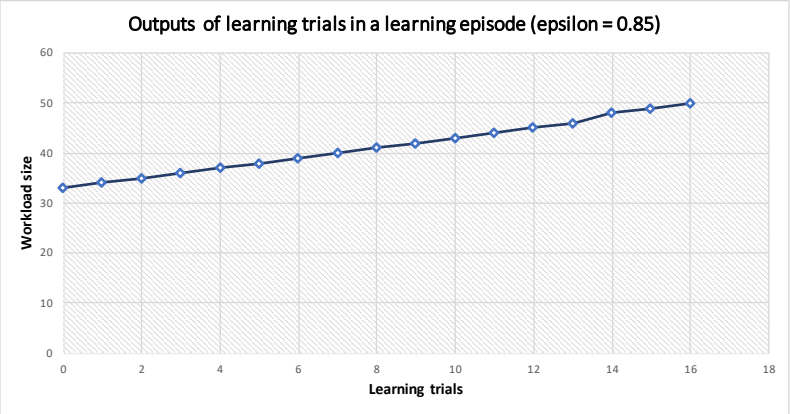


Figure 7.4: Output of learning trials in a learning episode

higher value such as 0.9, which causes faster updates in the stored Q-table of the agent, appears to work slightly better in adaptation to highly changeable environments. It leads to a smooth and partially equal performance during the learning episodes. Meanwhile, changing the values of the discount factor parameter does not show a considerable effect on the performance trend of the intelligent agent.

7.5 Related Work

Measuring performance metrics under different execution conditions including various workload and platform configurations [30, 31, 32], detecting different performance-related issues such as functional problems or violations of performance requirements which emerge under certain workload or resource configuration conditions [33, 34, 35] are often common objectives of different types of performance testing.

Different approaches to generate the test workload have been proposed in the literature. An overview of the used techniques for generating the test workload is presented as follows:

Analyzing system model. Analysis of a performance model of SUT in terms of Petri nets using constraint solving techniques [36], using genetic algorithms to

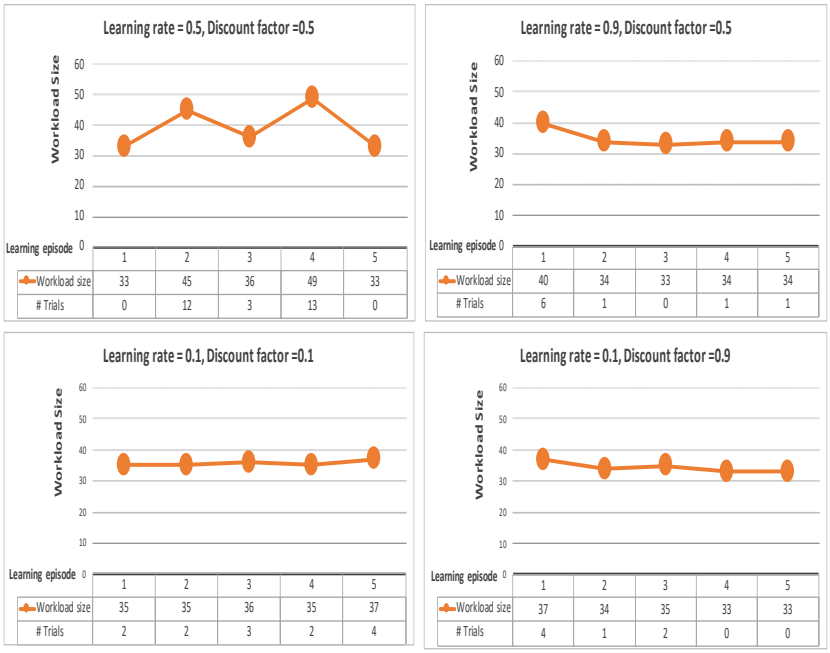


Figure 7.5: Behavioral sensitivity of the RL-assisted approach to the learning parameters

generate test load based on the control flow graph of SUT [15, 16], applying genetic-based algorithms to other types of system models such as UML models to generate stress test load [17, 18, 19, 20] are samples of the techniques proposed in this category.

Analyzing source code. Generating the test load using analysis of SUT data-flow and symbolic execution [37, 34] are examples of using source code analysis to generate test load and find performance-related issues.

Modeling real usage. Extracting the usage pattern of real users and modeling their behavior based on form-oriented models [21, 22], extracting workload characteristics and modeling the user behavior based on Extended Finite State Machines [38] and Markov chains [39] through monitoring submitted requests to SUT, and workload characterization through users clustering based on the business-level attributes extracted from usage data [40] are examples of the

techniques used for modeling the realistic workload.

Declarative methods. Using a declarative Domain Specific Language (DSL) to specify the performance testing process and a model-driven test execution framework [24, 25, 26], and also using a specific behavior-driven language, to specify load testing process in combination with a declarative performance testing framework like BenchFlow [23, 25] are examples of declarative techniques for performance and load testing.

Machine learning-assisted methods. Machine learning techniques such as supervised and unsupervised algorithms are often intended to build models and knowledge patterns from the data, while in other techniques like reinforcement learning algorithms, the intelligent agent learns the way to accomplish an objective through interaction with the environment. Machine learning techniques have been frequently used for analyzing the resulted data from the load testing, using Bayesian Network to predict the reliability from the load testing data [41], anomaly detection based on analysis of metrics data, e.g., resource usage, using clustering techniques [42], identifying performance signature based on performance metrics data using supervised and unsupervised learning techniques [43, 44] are some examples of using machine learning techniques for analysis of load testing data. Machine learning techniques have also been applied to the generation of performance test conditions in some studies. For example, RL together with symbolic execution has been applied to finding the worst-case execution path within a SUT in [45], and a feedback-driven learning technique which extracts some rules from the execution traces to find the performance bottlenecks, i.e., the method calls which their execution highly affects the performance of SUT [46]. Nonetheless, RL algorithms have been widely used in performance preservation of software services, such as an adaptive RL-driven performance control for cloud services [47, 48, 49] and also software services on other execution platforms [50, 51]. Regarding generating performance test conditions, a few studies have used some adaptive techniques, which are slightly analogous to the concept of RL, to generate the test workload. A feedback-based approach using search algorithms to benchmark an NFS server based on changing the test workload in [52], and an adaptive generation of test workload based on using some pre-defined adjusting policies in [35] are some examples related to the generation of performance test conditions.

7.6 Conclusion

User behavior models, system models, and source code are some common sources of information in load testing techniques for generating test workloads to find performance issues. Nonetheless, those artifacts might not be available all the time during the testing. In this paper, we proposed an adaptive model-free RL-driven load testing approach to accomplish the load testing objectives. We presented the architecture and operational procedure of the proposed smart load testing agent, which is able to learn the optimal policy to generate an efficient workload to reach the intended objective, without access to source code or system models. The agent reaches the intended objective with a fine-grained and smaller workload (in terms of the number of users) in comparison to a typical load testing process and results in time and cost savings in the testing process which is of great importance to many testing activities such as stress and regression load testing. The proposed approach uses Q-learning as the core learning mechanism with an adaptive action selection strategy to adapt the learning to further situations in the testing process. In other words, it learns the optimal policy and is also able to reuse it in subsequent similar testing scenarios, for example, reaching different values of error rate. Policy reuse results in further cost savings by reducing the required effort for workload generation.

In general, the generation of an efficient test workload while reducing dependency on source code, system and user behavior models and reduction of the required effort for workload generation in further situations, are the main strengths of the proposed RL-driven load testing. The value and benefit of the proposed intelligent load testing solution can particularly become more obvious in the provided benefits to stress and regression testing, and the possibility of testing independently of system and user behavior models which is beneficial to testing of complex and evolving software systems.

Acknowledgment

This work has been supported by and received funding partially from the TESTOMAT, XIVT, IVVES and MegaM@Rt2 European projects.

Bibliography

[1] Leonid Grinshpan. *Solving enterprise applications performance puzzles: queuing models to the rescue*. John Wiley & Sons, 2012.

[2] Andreas Brunnert, André van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Herbst, Pooyan Jamshidi, Reiner Jung, Joakim von Kistowski, et al. Performance-oriented devops: A research agenda. *arXiv preprint arXiv:1508.04752*, 2015.

[3] ISO 25000. ISO/IEC 25010 - System and software quality models, 2019. Available at https://iso25000.com/index.php/en/iso-25000-standards/iso-25010, Retrieved July, 2019.

[4] Martin Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26. IEEE, 2007.

[5] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.

[6] Zhen Ming Jiang and Ahmed E Hassan. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, 41(11):1091–1118, 2015.

[7] Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. Performance anomaly detection and bottleneck identification. *ACM Computing Surveys (CSUR)*, 48(1):4, 2015.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[9] Brendan Gregg. *Systems performance: enterprise and the cloud*. Pearson Education, 2013.

[10] Vittorio Cortellessa, Antinisca Di Marco, and Paola Inverardi. *Model-based software performance analysis*. Springer Science & Business Media, 2011.

[11] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[12] Krishna Kant and MM Srinivasan. *Introduction to computer system performance evaluation*. McGraw-Hill College, 1992.

[13] Giovanni Denaro, Andrea Polini, and Wolfgang Emmerich. Early performance testing of distributed software applications. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 94–103. ACM, 2004.

[14] Pingyu Zhang, Sebastian Elbaum, and Matthew B Dwyer. Compositional load test generation for software pipelines. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 89–99. ACM, 2012.

[15] Yuanyan Gu and Yujia Ge. Search-based performance testing of applications with composite services. In *2009 International Conference on Web Information Systems and Mining*, pages 320–324. IEEE, 2009.

[16] Massimiliano Di Penta, Gerardo Canfora, Gianpiero Esposito, Valentina Mazza, and Marcello Bruno. Search-based testing of service level agreements. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1090–1097. ACM, 2007.

[17] Vahid Garousi. A genetic algorithm-based stress test requirements generator tool and its empirical evaluation. *IEEE Transactions on Software Engineering*, 36(6):778–797, 2010.

[18] Vahid Garousi, Lionel C Briand, and Yvan Labiche. Traffic-aware stress testing of distributed real-time systems based on uml models using genetic algorithms. *Journal of Systems and Software*, 81(2):161–185, 2008.

[19] Leandro T Costa, Ricardo M Czekster, Flávio Moreira de Oliveira, Elder de M Rodrigues, Maicon Bernardino da Silveira, and Avelino F Zorzo. Generating performance test scripts and scenarios based on abstract intermediate models. In *SEKE*, pages 112–117, 2012.

[20] Maicon Bernardino da Silveira, Elder de M Rodrigues, Avelino F Zorzo, Leandro T Costa, Hugo V Vieira, and Flávio Moreira de Oliveira. Generation of scripts for performance testing based on uml models. In *SEKE*, pages 258–263, 2011.

[21] Dirk Draheim, John Grundy, John Hosking, Christof Lutteroth, and Gerald Weber. Realistic load testing of web applications. In *Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 11–pp. IEEE, 2006.

[22] Christof Lutteroth and Gerald Weber. Modeling a realistic workload for performance testing. In *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 149–158. IEEE, 2008.

[23] Henning Schulz, Dušan Okanović, André van Hoorn, Vincenzo Ferme, and Cesare Pautasso. Behavior-driven load testing using contextual knowledge-approach and experiences. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pages 265–272. ACM, 2019.

[24] Vincenzo Ferme and Cesare Pautasso. A declarative approach for performance tests execution in continuous software development environments. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 261–272. ACM, 2018.

[25] Vincenzo Ferme and Cesare Pautasso. Towards holistic continuous software performance assessment. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, pages 159–164. ACM, 2017.

[26] Jürgen Walter, Andre van Hoorn, Heiko Koziolek, Dusan Okanovic, and Samuel Kounev. Asking what?, automating the how?: The vision of declarative performance engineering. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, pages 91–94. ACM, 2016.

[27] Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. Machine learning to guide performance testing: An autonomous test framework. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 164–167. IEEE, 2019.

[28] Apache. JMeter. Available at https://https://jmeter.apache.org/, Retrieved October, 2019.

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[30] Daniel A Menascé. Load testing, benchmarking, and application performance management for the web. In *Int. CMG Conference*, pages 271–282, 2002.

[31] Varsha Apte, TVS Viswanath, Devidas Gawali, Akhilesh Kommireddy, and Anshul Gupta. Autoperf: Automated load testing and resource usage profiling of multi-tier internet applications. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pages 115–126. ACM, 2017.

[32] Anshul Jindal, Vladimir Podolskiy, and Michael Gerndt. Performance modeling for cloud microservice applications. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pages 25–32. ACM, 2019.

[33] Lionel C Briand, Yvan Labiche, and Marwa Shousha. Stress testing real-time systems with genetic algorithms. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1021–1028. ACM, 2005.

[34] Pingyu Zhang, Sebastian Elbaum, and Matthew B Dwyer. Automatic generation of load tests. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 43–52. IEEE Computer Society, 2011.

[35] Vanessa Ayala-Rivera, Maciej Kaczmarski, John Murphy, Amarendra Darisa, and A Omar Portillo-Dominguez. One size does not fit all: In-test

workload adaptation for performance testing of enterprise applications. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 211–222. ACM, 2018.

[36] Jian Zhang and Shing Chi Cheung. Automated test case generation for the stress testing of multimedia systems. *Software: Practice and Experience*, 32(15):1411–1435, 2002.

[37] Cheer-Sun D Yang and Lori L Pollock. Towards a structural load testing tool. In *ACM SIGSOFT Software Engineering Notes*, volume 21, pages 201–208. ACM, 1996.

[38] Mahnaz Shams, Diwakar Krishnamurthy, and Behrouz Far. A model-based approach for testing the performance of web applications. In *Proceedings of the 3rd international workshop on Software quality assurance*, pages 54–61. ACM, 2006.

[39] Christian Vögele, André van Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. Wessbas: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems. *Software & Systems Modeling*, 17(2):443–477, 2018.

[40] Gururaj Maddodi, Slinger Jansen, and Rolf de Jong. Generating workload for erp applications through end-user organization categorization using high level business operation data. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 200–210. ACM, 2018.

[41] Alberto Avritzer, Flávio P Duarte, Rosa Maria Meri Leao, Edmundo de Souza e Silva, Michal Cohen, and David Costello. Reliability estimation for large distributed software systems. In *Cascon*, page 12. Citeseer, 2008.

[42] Mark D Syer, Bram Adams, and Ahmed E Hassan. Identifying performance deviations in thread pools. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 83–92. IEEE, 2011.

[43] Haroon Malik, Hadi Hemmati, and Ahmed E Hassan. Automatic detection of performance deviations in the load testing of large scale systems.

In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1012–1021. IEEE Press, 2013.

[44] Haroon Malik, Zhen Ming Jiang, Bram Adams, Ahmed E Hassan, Parminder Flora, and Gilbert Hamann. Automatic comparison of load tests to support the performance analysis of large enterprise systems. In *2010 14th European conference on software maintenance and reengineering*, pages 222–231. IEEE, 2010.

[45] Jinkyu Koo, Charitha Saumya, Milind Kulkarni, and Saurabh Bagchi. Pyse: Automatic worst-case test generation by reinforcement learning. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 136–147. IEEE, 2019.

[46] Mark Grechanik, Chen Fu, and Qing Xie. Automatically finding performance problems with feedback-directed learning software testing. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 156–166. IEEE, 2012.

[47] Olumuyiwa Ibidunmoye, Mahshid Helali Moghadam, Ewnetu Bayuh Lakew, and Erik Elmroth. Adaptive service performance control using cooperative fuzzy reinforcement learning in virtualized environments. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 19–28. ACM, 2017.

[48] T Veni and S Mary Saira Bhanu. Auto-scale: automatic scaling of virtualised resources using neuro-fuzzy reinforcement learning approach. *International Journal of Big Data Intelligence*, 3(3):145–153, 2016.

[49] Pooyan Jamshidi, Amir Sharifloo, Claus Pahl, Hamid Arabnejad, Andreas Metzger, and Giovani Estrada. Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. In *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*, pages 70–79. IEEE, 2016.

[50] Mahshid Helali Moghadam and Seyed Morteza Babamir. Makespan reduction for dynamic workloads in cluster-based data grids using reinforcement-learning based scheduling. *Journal of computational science*, 24:402–412, 2018.

[51] Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. Adaptive runtime response time control in PLC-based real-time systems using reinforcement learning. In *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 217–223. IEEE, 2018.

[52] Piyush Shivam, Varun Marupadi, Jeffrey S Chase, Thileepan Subramaniam, and Shivnath Babu. Cutting corners: Workbench automation for server benchmarking. In *USENIX Annual Technical Conference*, pages 241–254, 2008.

Chapter 8

Paper D: Adaptive Runtime Response Time Control in PLC-Based Real-Time Systems Using Reinforcement Learning

Mahshid Helali Moghadam, Mehrdad Saadatmand,
Markus Borg, Markus Bohlin, Björn Lisper
In the Proceedings of the 13th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2018), Gothenburg, Sweden, May 2018.

Abstract

Timing requirements such as constraints on response time are key characteristics of real-time systems and violations of these requirements might cause a total failure, particularly in hard real-time systems. Runtime monitoring of the system properties is of great importance to check the system status and mitigate such failures. Thus, a runtime control to preserve the system properties could improve the robustness of the system with respect to timing violations. Common control approaches may require a precise analytical model of the system which is difficult to be provided at design time. Reinforcement learning is a promising technique to provide adaptive model-free control when the environment is stochastic, and the control problem could be formulated as a Markov Decision Process. In this paper, we propose an adaptive runtime control using reinforcement learning for real-time programs based on Programmable Logic Controllers (PLCs), to meet the response time requirements. We demonstrate through multiple experiments that our approach could control the response time efficiently to satisfy the timing requirements.

8.1 Introduction

Real-time control programs implemented on Programmable Logic Controllers (PLCs) are key parts of many time-critical industrial control systems like those in the railway domain. The timing properties in these systems include period of tasks, deadline, worst-case execution time or response time. From the perspective of timing analysis, schedulability analysis methods, statistical and formal timing analysis are common analysis techniques to provide a response time estimation of real-time programs [1, 2, 3]. Static analysis-based approaches, in some cases, might not be practical for complex real-time systems. Even if they are feasible, the results might not be valid due to unpredictable factors in runtime and the difference between analysis environment and the realistic one [4].

Generally, there is often a strict set of timing requirements such as deadlines and limits on response time for real-time programs in mission-critical contexts. Correctness of functionality of real-time systems highly depends on satisfying the timing requirements as important features of these systems. Any serious deviation in temporal behavior of real-time programs due to unpredicted runtime events like asynchronous message-passing and runtime changeable priorities, particularly in complex systems, might cause a total failure in the function of system. Thus, providing more robustness against unpredicted varying conditions during runtime is of great importance. In general, robustness could be defined as to which degree the system is tolerable against incorrect inputs or unexpected stressed conditions [5]. In a real-time program, robustness could be defined as the ability to adapt to the varying conditions while satisfying the timing requirements.

An adaptive runtime control in addition to the scheduling capabilities could lead to more robustness in real-time control systems, to cope with changing runtime conditions and unpredicted states [6]. Runtime monitoring could check if the system adheres to the predefined requirements like timing constraints. A control approach based on runtime monitoring could help preserve these timing properties by applying runtime control operations. Adaptive control strategies are considered as one of the promising solutions to improve robustness through providing adaptation to the varying conditions in dynamic environments. Reinforcement learning (RL) has been frequently applied to address the adaptive control strategy in dynamic environments, in case the envi-

ronment is stochastic, and the control problem can be formulated as a Markov Decision Process (MDP).

In this paper, we propose a self-adaptive response time control for real-time programs in PLC-based systems using reinforcement learning. In our previous work [7], we presented the initial idea on how a learning-based solution can be used to provide assurance of timing properties; here in this work we extend that initial idea and provide an industrial evaluation of our proposed approach. We present the evaluation experiments of the proposed approach on sample programs inspired from our collaboration with Bombardier Transportation in Sweden. The proposed approach formulates the response time control problem as an MDP and uses Q-learning as a model-free RL to provide adaptive control of response time while meeting the timing requirement. We show the efficacy of the proposed approach through multiple experiments based on simulating real-time programs in a PLC-based control system. Our approach mostly keeps the programs adhering to the response time constraints despite the occurred time deviations during the run time. Based on the evaluation results, the proposed approach with ε-greedy, $\varepsilon = 0.5$, and $\alpha = 0.1$ and $\gamma = 0.5$ provided better satisfaction of the response time threshold without any programs ending with medium or high deviation.

The rest of this paper is organized as follows; Section 8.2 discusses briefly the motivation and background concepts of RL. The technical details of the proposed approach are discussed in Section 8.3, while Section 8.4 presents the evaluation experiments and results. Section 8.5 provides a review of the related works and background techniques. Conclusions and future directions are provided in Section 8.6.

8.2 Motivation and Background

8.2.1 Motivation

Runtime monitoring is considered as a principal means for real-time systems. Providing an adaptive control for satisfying the timing requirements such as constraints on response time/execution time based on runtime monitoring could improve the robustness of the system. Model-driven control approaches may require precise knowledge of the system and environment. The complexity of real-time systems, for example, intricate temporal dependencies between

real-time tasks and the dynamism of the environment are major barriers which motivate towards model-free learning-based control. Learning-based control can find an adaptive control policy to varying conditions regardless of having a precise model of the environment. Reinforcement learning-based control techniques have been used for runtime control of non-functional properties to satisfy the performance and timing requirements in many application contexts.

Reinforcement learning [8] is a learning mechanism working based on interaction with the environment. In RL, the agent senses the state of the environment continuously, takes a possible action and in return, receives a reward signal from the environment which shows the desirability and effectiveness of the applied action. During the learning, the agent follows a policy which maximizes the long-term received reward. The agent learns this policy through an action selection strategy which is based on selecting an action randomly (exploration) or selecting an action with a high utility value (exploitation). Q-learning [8] is a model-free RL algorithm in which the agent learns the value function of the long-term expected reward associated to the pairs of states and actions. It is an off-policy learning as the optimal policy is learnt independently of the action selection strategy being used by the agent. Once the learning converges, the agent replays the learned policy.

8.2.2 PLC-Based Industrial Control Programs

Many of the real-time industrial control systems like those ones in the transportation domain, are implemented based on IEC 61131-3 [9] which is one of the main programming language standards for programmable controllers. According to the proposed software structure in IEC 61131-3, Programmable Organization Units (POU) are the building blocks of a PLC program. They are hardware-independent and programmable in a flexible fashion facilitating the reusability and modularization in this context.

There are mainly three unified types of POUs: program, function block and function. A function block has its own data record to remember the state of the information, while a function always produces the same result based on the same input. A program may consist of zero or multiple function and function blocks. A real-time task can execute one or multiple programs or a set of function blocks. Timer function blocks are widely used as one of the main constituent POUs in PLC-based real-time programs. Their basic functions involve

providing their output after a preset controllable/programmable time interval. There are three types of timers as standard PLC timer blocks, i.e., TP (Timer Pulse), TON (Timer On-Delay) and TOF (Timer Off-Delay). Timer TP is a pulse generator which supplies a constant pulse on output upon detecting a rising edge at input. TON supplies the value of input at output with a delay upon detecting a rising edge at input. TOF has an inverse functionality to TON. Figure 8.1 shows a schema sample from a real-time control program in Function Block Diagram format, as an integration of multiple functions and function blocks. The number of POUs in each control program depends on the complexity of the program. The time delay of timer function blocks in time-critical programs are the target entities supposed to be tuned in urgent conditions by our control approach to satisfy the response time requirements.

8.3 Adaptive Response Time Control Using Q-Learning

In this section, we present the technical details of the proposed runtime response time control using reinforcement learning for real-time programs running on PLC-based systems. This control method is incorporated into the control scan program which is responsible for executing the building blocks and preserving their execution orders within real-time programs. Timer function blocks are one of the standard function blocks which are widely used and play a key role in many time-critical industrial control programs.

The proposed control strategy is supposed to use the capability of tuning the time delay of timer function blocks to control the response time of real-time programs. The main objective of the proposed runtime response time

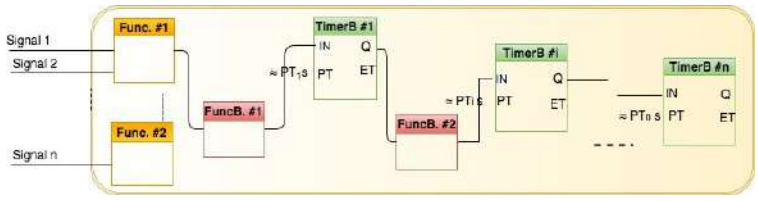


Figure 8.1: A schema sample from a PLC-based control program

control is to meet the response time requirements in abnormal conditions when time deviations happen, by optimally tuning the time delay parameters. The proposed approach uses Q-Learning as a model-free RL to learn the optimal tuning of delay parameters to preserve the program responsive within the target response time threshold. The learning task in the proposed control approach mainly involves the following steps:

1) Detecting the *State* of the system. Based on the interactive characteristic of the reinforcement learning, the control agent/controller observes the state of the program at discrete time steps. After each execution cycle, the controller measures the execution time until the current time point. The actual execution time until the end of the n^{th} function block execution, ET_n, is classified under four classes. This is done based on the amount of compliance with the desired/target execution time until the end of the nth function block (e.g., from requirements/constraints), T_n, calculated as follows:

$$T_n = \sum_{i=1}^{n} T_i^f \tag{8.1}$$

Where T_i^f is the desired response time of the i^{th} function block. The class values representing the state of the program, s, are *Required, Low, Medium* and *High*, as shown in Figure 8.2. They represent the acceptable state, and the states with low, medium and high deviation, respectively. We defined the acceptable state based on a target execution time characterized by a tolerance region $[T_n, T'_n]$ where $T'_n = T_n + \tau$. where τ in *ms* is defined based on the characteristics of the system.

2) Selecting a *Control Action*. We defined the control actions as tuning operations for the time delay of the next running function block, D_f^{n+1}. For providing a safety margin, we also considered a required minimum delay, D_m, for function blocks. Then, the time delay of function blocks could not be set to a value less than D_m. Regarding the minimum time delay, we specified a set of control actions for tuning the time delay as follows:

$$Actions = \{(1 - f_d)D_f^{n+1} + f_d D_m : f_d \in K\} \tag{8.2}$$

$$K = \{0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1\} \tag{8.3}$$

Where f_d is a decreasing factor.

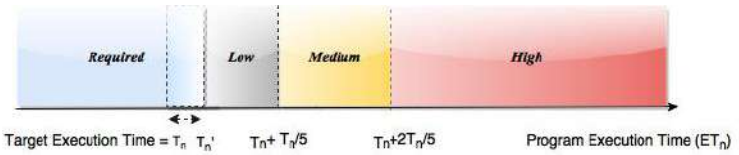


Figure 8.2: States of the program

3) Receiving the *Reward* signal and updating the stored experience. After applying the selected action, the system will go to the next state and the controller will receive a reward signal representing the effectiveness of the applied action. We derived a utility function based on a Normal probability density function with $\mu = T_n$ and $\sigma = T_n/10$ which is as follows:

$$r_n = \begin{cases} \frac{1}{\frac{T_n}{10}\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{ET_n - T_n}{\frac{T_n}{10}})^2}, & T_n < ET_n \\ 1, & ET_n \leq T_n \end{cases} \tag{8.4}$$

The computed reward values will be in the range (0, 1].

The final objective of the learning is to find a policy π, a mapping between the states and actions, which maximizes the expected long-term reward defined as follows [8]:

$$R_n = r_{n+1} + \gamma_{n+2} + ... + \gamma^k r_{n+k+1} \tag{8.5}$$

Where $\gamma \in [0, 1]$ is a discount factor specifying the importance of future rewards compared to the immediate reward. The long-term expected return of selecting action *a* in state *s*, based on policy π, is specified by a utility value $Q^\pi(s, a)$ defined as follows [8]:

$$Q^\pi(s, a) = E^\pi[R_n | S_n = s, A_n = a] \tag{8.6}$$

The Q-values stored in a look-up table, *Q-table*, form the experience of the agent. The controller relies on Q-values to make decision on actions. During the learning, the Q-values are updated incrementally via temporal differencing. The agent updates the associated $Q^\pi(s, a)$ for each experienced (s, a) through the following rule:

$$Q(s_n, a_n) = Q(s_n, a_n) + \alpha[r_{n+1} + \gamma \max_a Q(s_{n+1}, a) - Q(s_n, a_n)] \tag{8.7}$$

Where $\alpha \in [0, 1]$ is the learning rate parameter. It specifies to what extent new information impacts the q-values. The all steps of the adaptive control procedure are described in Algorithm 10. Eventually, after multiple learning cycles, the controller finds the optimal policy of selecting the action which maximizes the q-value in a given state.

Learning performance. Different action selection strategies could be used during the learning. The agent can use a random action selection method or select greedily an action with the highest utility value according to the Q-table. ε-*greedy* is an action selection strategy which allows the agent to make a trade-off between the exploration and exploitation in the action space. In ε-greedy, with probability ε, a random action is selected and with probability $1 - \varepsilon$, an action based on the utility value is selected. However, RL-based approaches might generally suffer slow convergence due to the need for exploring the state space. To alleviate this effect, we also introduced an initial control mapping in Q-table by specifying some invalid pairs of state and action to guide the agent not to explore specific actions in a specific state. For example, when it is in acceptable state, no need to change the time delay parameter.

Algorithm 10 Adaptive Response Time Control in PLC-based Real-time programs

Required: S, A, $\varepsilon, \alpha, \gamma, \phi$ (invalid state-action pairs)
Initialize Q-values, $Q(s, a) = -1 \; if \; (s, a) \in \phi \; else \; 0 \; \forall s \in S, \; \forall a \in A$
1. Detect the current state of the program, s_n
2. Select an action 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)
3. Apply the selected action, let the system continue running and execute the next function block
4. Detect the new state of the system
5. Compute the reward (reinforcement) signal
6. Update the Q-value by:
$Q(s_n, a_n) = Q(s_n, a_n) + \alpha[r_{n+1} + \gamma \max_a Q(s_{n+1}, a) - Q(s_n, a_n)]$
7. Repeat for every observed state at the start of each function block execution

8.4 Results and Discussion

This section presents the results of the early stage evaluation experiments addressing the performance of the proposed approach in terms of meeting the predefined response time threshold. The main objective of the experiments is to assess to which degree the learning-based control can work adaptively on varying conditions and untimely behavior of function blocks in a realistic environment.

8.4.1 Evaluation Setup

In this study, we implemented the proposed approach based on three action selection strategies. We incorporated it into an environment which simulates multiple real-time programs consisting of various timer function blocks. The simulation environment emulates the temporal behavior of the function blocks, their responses in realistic environments and the corresponding control scan program for controlling the execution order of the function blocks. The learning-based control has been integrated into the control scan thread to provide a run-time control of the response time of real-time programs.

The proposed approach has been evaluated through two analysis scenarios. In the first scenario, concerning response time analysis, the performance of the learning-based control based on using three action selection algorithms has been studied. In this scenario, the performance of the proposed approach after 100 learning episodes (interaction with various real-time programs) has been demonstrated. The real-time programs have been characterized with different numbers of function blocks, predefined response time requirements and minimum required delay time (safety margin). The second analysis scenario, sensitivity analysis, analyzes the sensitivity of the learning-based approach to the learning parameters. This scenario involves investigating the effects of the learning parameters by systematically changing the values of one parameter while keeping the other one constant.

8.4.2 Experiments and Results

Timing Analysis. In the timing analysis scenario, the efficacy of the learning-based approach was evaluated in terms of adaptation to changeable behavior while meeting the timing requirement. The simulated real-time programs have

different numbers of function blocks in the range [5, 25]. The predefined response time requirements of function blocks and associated safety margins in *ms* have been initialized with values in the range [1000, 6000] and [1000, 2000], respectively. A maximum deviation at most equal to 25 percent of the upper bound of the response time requirement was allowed during the simulation. The default acceptable tolerance value was considered as 500*ms*. Time deviations were injected into the programs randomly. Figure 8.3 shows a random pattern for injecting time deviations to function blocks within three program samples. ε-greedy was used in the proposed approach as an action selection strategy with $\varepsilon = 0.1$, $\varepsilon = 0.5$ and $\varepsilon = 0.9$. The ε-value determines to what extent exploration and exploitation are weighted during the action selection procedure. Figures 8.4 and 8.5 show the observed response time plots of real-time programs after applying the learning-based control approach based on different values of ε parameter in the action selection strategy. Clearly, the learning-based control approach tries to adapt well to the varying temporal behaviors of the function blocks while meeting the response time thresholds of the programs. Results in Figures 8.4 and 8.5 describe the efficacy of the learning-based control approach based on the number of programs ended with medium or high deviations from the timing requirements and also the achieved average deviations.

According to the results, the performance of the proposed approach with different action selection strategies is described as follows:

1) ε-greedy with $\varepsilon = 0.1$ makes the controller trust most on its stored experience, rather than exploring new actions. The learning-based approach based

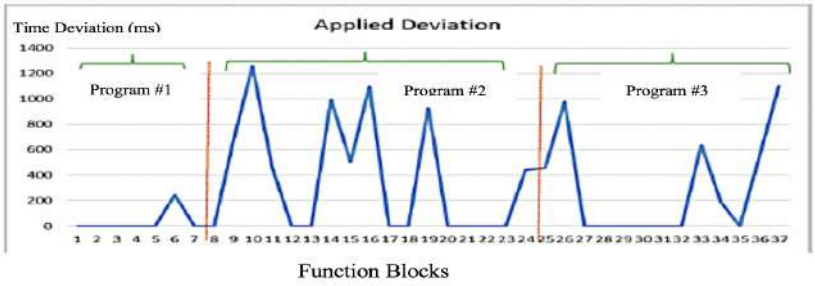


Figure 8.3: Pattern of time deviations

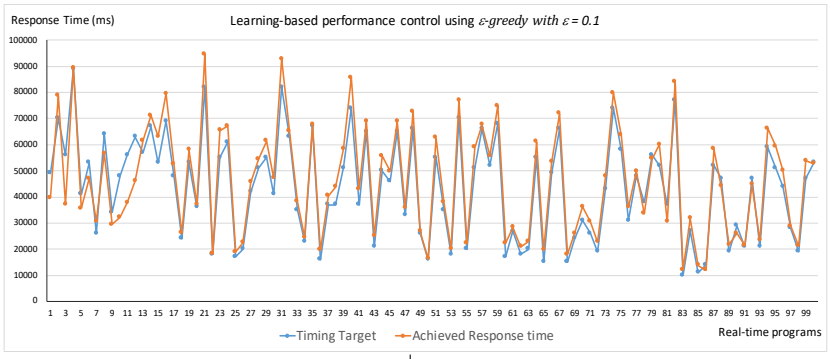
on this action selection strategy, showed less efficiency in terms of optimizing the response time and also the number of programs which ended with medium or high deviations. In this case, the experience of the controller has not been extended well and needs more exploration to be improved.

2) ε-greedy with $\varepsilon = 0.9$ provides more opportunities towards the exploration of the action space. It provided partially better performance in terms of optimizing the response time and preventing the programs from exceeding the predefined thresholds with medium or high deviations compared to ε-greedy with ε= 0.1.

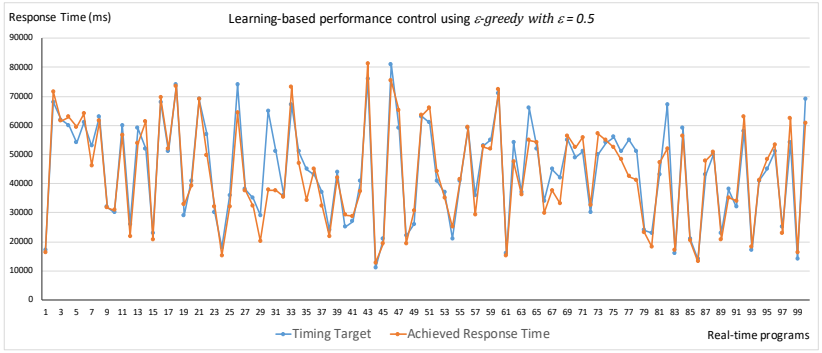
3) ε-greedy with $\varepsilon = 0.5$ provides a trade-off between the exploration and exploitation of the action space. It showed a better adaptation to the varying conditions and tried to preserve the response time close to the requirement threshold. In some cases where a sharp satisfaction of the timing requirement is needed, e.g., airbag control systems of automotive products, this is the desired performance which is required.

4) ε-greedy with decaying ε, is an action selection strategy during which the ε parameter gradually decreases. It causes more exploration during the first steps of the learning and more exploitation at the last steps. Using this strategy, the performance controller first explores the action space, then tends towards using the achieved experience. The learning-based approach based on ε-greedy with decaying ε, showed the most promising results, i.e., it outperformed the other ε-strategies both in terms of optimizing response time and preventing medium or high deviations from the predefined timing thresholds.

Sensitivity Analysis. The behavior of the proposed learning-based control approach could be impacted by the learning parameters including learning rate (α) and discount factor (γ). In the sensitivity analysis, two sets of experiments were done to study the effects of varying learning parameters. Each set of experiments involves changing the value of one parameter while keeping the other one constant. ε-greedy with $\varepsilon = 0.5$ was used as a baseline action selection strategy during the sensitivity analysis experiments. Table 8.1 shows the performance of the learning-based approach regarding the number of real-time programs which ended with medium or high deviations from the predefined response time thresholds and also the achieved average deviation in response time, during the sensitivity analysis experiments. In Table 8.1 the bold column represents the baseline parameter setting which was used in each sensitivity analysis experiment. We set the learning rate to 0.1 and the discount factor to

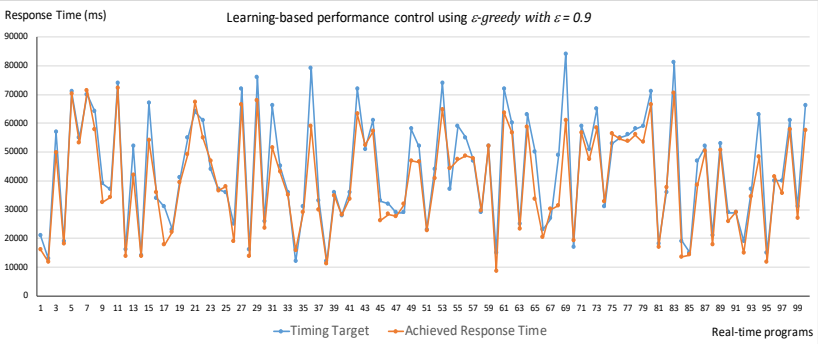


Average injected time deviation per program: 5504 ms	*LR-based using ε-greedy, ε= 0.1*	*Uncontrolled*
#RT programs with highly exceeded predefined threshold	6	13
Achieved average deviation	2585	5504

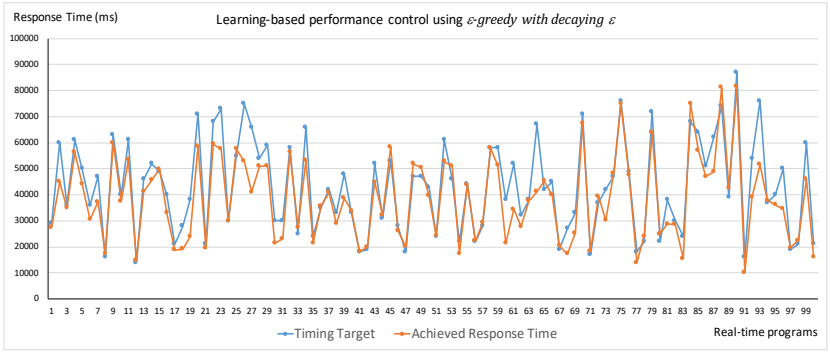


Average injected time deviation per program: 5731 ms	*LR-based using ε-greedy, ε= 0.5*	*Uncontrolled*
#RT programs with highly exceeded predefined threshold	0	10
Achieved average deviation	-1228	5731

Figure 8.4: response time plots of real-time programs and the performance of the adaptive learning-based control



Average injected time deviation per program: 5911 ms	*LR-based using ε-greedy, ε= 0.9*	*Uncontrolled*
#RT programs with highly exceeded predefined threshold	1	7
Achieved average deviation	-3804	5911



Average injected time deviation per program: 5395 ms	*LR-based using decaying ε-greedy*	*Uncontrolled*
#RT programs with highly exceeded predefined threshold	0	8
Achieved average deviation	-4531	5395

Figure 8.5: response time plots of real-time programs and the performance of the adaptive learning-based control

0.5 at the first and second experiments, respectively.

It seems that setting the learning rate to 0.1, which provides a slower learning, leads to good performance, particularly in adaptation to varying behaviors and preventing the real-time programs from exceeding the timing thresholds. Increasing the learning rate towards 0.5, which aims at balancing between learning new information and saving previous experience, causes improvement in optimizing the response time of the programs. The proposed approach also does not show as much performance improvement as when we set the discount factor to values other than 0.5.

8.5 Related Work

We classify the relevant works on timing properties of real-time systems under modeling, verifying and some approaches to preserve and satisfy the timing requirements. Many of the verification and preservation/control approaches are based on runtime monitoring of the properties. Real-time Specification for Java (RTSJ) was introduced to provide a real-time scheduler with the facility of monitoring deadlines and enforcing the execution cost [10]. Mezzetti et al [11] used the Ada Ravenscar Profile for preserving the timing properties of real-time systems. Saadatmand et al [12, 13] developed an extra scheduler taking the temporal properties including period, execution time and deadline of the tasks and scheduled them using the underlying scheduler of the operating system. A model synthesis approach for timing properties of real-time systems based on monitoring the running system was proposed in [14]. A runtime framework for monitoring the runtime constraints such as timing constraints and detecting the violations of timing properties was presented in [15]. The related issues on runtime monitoring of properties in real-time systems were discussed in [16]. Goodloe et al [6] surveyed different runtime monitoring techniques including off-line and on-line techniques for distributed real-time systems, in particular hard real-time systems. Das et al [17] presented a tool environment which provided runtime monitoring, animating the development and analysis of the components to support model-driven development of real-time embedded systems. In [18] a runtime monitoring approach for checking the system properties in embedded systems was presented. It used a control method to coordinate the time predictability and memory utilization in the monitoring solution.

Table 8.1: Impacts of varying learning parameters on the performance of control approach

	LR-based performance control using ε= 0.5, Discount factor γ=0.5			*LR-based performance control using ε= 0.5, Learning rate α=0.1*		
	α= **0.1**	α=0.5	α=0.9	γ=0.1	γ=**0.5**	γ=0.9
#RT programs with highly exceeded predefined threshold (Uncontrolled Condition)	**0(10)**	0(6)	0(3)	3(8)	**0(10)**	0(7)
Average deviation (Uncontrolled Condition)	**-1228 (5731)**	-6475 (5378)	-4395 (5455)	-1052 (5484)	**-1228 (5731)**	-1210 (5744)

8.6 Conclusion

Runtime monitoring of system properties remains as a principal need for real-time systems. A runtime control approach based on runtime monitoring could improve robustness of the system. In this paper, we present an adaptive runtime response time control based on reinforcement learning for PLC-based real-time programs, to satisfy the timing requirements. In this study, we formulate the control problem as an MDP and apply Q-learning to provide a control technique to preserve the response time according to the timing requirements. We evaluate the efficacy of the approach through multiple experiments. The learning-based approaches generally require multiple learning trials to converge and stabilize the learned policy. Regarding this issue and the characteristics of soft and hard real-time systems, it is supposed that the proposed learning-based approach in its incremental learning fashion could be used in soft real-time systems. While the controller with the converged policy, after training based on simulation environment, could be integrated into the hard real-time systems. Furthermore, the result values (the tuned values) of the control policy could be used as a feedback to correct the initial model of the system. Future directions of this study will be evaluating the efficacy of the approach in the industrial platforms, improving the training time and adaptation

precision of the approach by modeling the state space as fuzzy state space and using cooperative agents to speed up the learning.

Acknowledgment

This research has been funded by the ITEA3 initiative TESTOMAT Project (www.testomatproject.eu) through Vinnova and by KKS through the TOCSYC project).

Bibliography

[1] Giordano A Kaczynski, Lucia Lo Bello, and Thomas Nolte. Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 101–110. IEEE, 2007.

[2] Sorin Manolache, Petru Eles, and Zebo Peng. Schedulability analysis of applications with stochastic task execution times. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(4):706–735, 2004.

[3] Elena Fersman, Pavel Krcal, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149–1172, 2007.

[4] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. Design of adaptive security mechanisms for real-time embedded systems. In *International Symposium on Engineering Secure Software and Systems*, pages 121–134. Springer, 2012.

[5] ANSI/IEEE. Standard glossary of software engineering terminology. 1991.

[6] Alwyn E Goodloe and Lee Pike. Monitoring distributed real-time systems: A survey and future directions. 2010.

[7] Mahshid Helali Moghadam, Mehrdad Saadatmand, Markus Borg, Markus Bohlin, and Björn Lisper. Learning-based self-adaptive assurance of timing properties in a real-time embedded system. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 77–80. IEEE, 2018.

[8] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[9] IEC Iec. 61131-3: Programmable controllers–part 3: Programming languages. *International Standard, Second Edition, International Electrotechnical Commission, Geneva*, 1:2003, 2003.

[10] Andy Wellings, Gregory Bollella, Peter Dibble, and David Holmes. Cost enforcement and deadline monitoring in the real-time specification for java. In *Seventh IEEE International Symposium onObject-Oriented Real-Time Distributed Computing, 2004. Proceedings.*, pages 78–85. IEEE, 2004.

[11] Enrico Mezzetti, Marco Panunzio, and Tullio Vardanega. Preservation of timing properties with the ada ravenscar profile. In *International Conference on Reliable Software Technologies*, pages 153–166. Springer, 2010.

[12] Mehrdad Saadatmand, Mikael Sjödin, and Naveed Ul Mustafa. Monitoring capabilities of schedulers in model-driven development of real-time systems. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pages 1–10. IEEE, 2012.

[13] Nima Asadi, Mehrdad Saadatmand, and Mikael Sjödin. Run-time monitoring of timing constraints: A survey of methods and tools. In *the Eighth International Conference on Software Engineering Advances*, 2013.

[14] Joel Huselius and Johan Andersson. Model synthesis for real-time systems. In *Ninth European Conference on Software Maintenance and Reengineering*, pages 52–60. IEEE, 2005.

[15] Farnam Jahanian. Run-time monitoring of real-time systems. *Advances in Real-Time Systems. Prentice Hall*, 1995.

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

[17] Nondini Das, Suchita Ganesan, Leo Jweda, Mojtaba Bagherzadeh, Nicolas Hili, and Juergen Dingel. Supporting the model-driven development

of real-time embedded systems with run-time monitoring and animation via highly customizable code generation. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 36–43. ACM, 2016.

[18] Ramy Medhat, Borzoo Bonakdarpour, Deepak Kumar, and Sebastian Fischmeister. Runtime monitoring of cyber-physical systems under timing and memory constraints. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(4):79, 2015.

Chapter 9

Paper E: Makespan Reduction for Dynamic Workloads in Cluster-Based Data Grids Using Reinforcement Learning Based Scheduling

Mahshid Helali Moghadam, Seyed Morteza Babamir
Journal of Computational Science, Vol. 24, January 2018.

Abstract

Scheduling is one of the important problems within the scope of control and management in grid and cloud-based systems. Data grid still as a primary solution to process data-intensive tasks, deals with managing large amounts of distributed data in multiple nodes. In this paper, a two-phase learning-based scheduling is proposed for data-intensive tasks scheduling in cluster-based data grids. In the proposed approach, a hierarchical multi agent system, consisting of one global broker agent and several local agents, is applied to scheduling procedure in the cluster-based data grids. At the first step of the proposed approach, the global broker agent selects the cluster with the minimum data cost based on the data communication cost measure, then an adaptive policy based on Q-learning is used by the local agent of the selected cluster to schedule the task to the proper node of the cluster. The impacts of three action selection strategies have been investigated in the proposed approach, and the performance of different versions of the approach regarding different action selection strategies, has been evaluated under three types of workloads with heterogeneous tasks. Experimental results show that for dynamic workloads with varying task submission patterns, the proposed learning-based scheduling gives better performance compared to four common scheduling strategies, Queue Length (Shortest Queue), Access Cost, Queue Access Cost (QAC) and HCS, which use regular combinations of primary parameters such as, data communication cost and queue length. Applying a learning-based strategy provides the scheduling with more adaptability to the changing conditions in the environment.

9.1 Introduction

Grid computing is a distributed computing system, which enables the integrated and collaborative use of many heterogeneous resources owned by different organizations. It is still used for computation- and data-intensive processing. Data grid deals primarily with data-intensive applications. Many scientific and engineering problems require to access and process large amounts of distributed data [1, 2, 3]. In many application environments, there is a complex of heterogeneous tasks, which are quite different in terms of their sizes and their processing requirements. Studies show that the variability of task sizes is an important factor, which highly affects the performance of the scheduling [4]. Using adaptive control strategies in dynamic environments with varying conditions may be a proper solution to deal with changing features of the problem environment. Adaptive control is a type of control dealing with time-varying parameters. It does not need a priori knowledge about the uncertain parameters and involves a control method changing itself.

Learning automata is a machine-learning field considered as an adaptive control method [5]. Generally, in learning automata, the current action is selected based on the experiences collected from the environment. It may be in the domain of reinforcement learning (RL), if the environment is stochastic and can be modelled as a Markov Decision Process (MDP). In a grid system, the state of the system can be described by a random process, X_{t_n}, specifying the state of the system as a function of time. The state space of the system is known, and the scheduling of submitted tasks is conducted according to the current state of the system. Therefore, the random process describing the system is a Markov process and satisfies the Markov property as follows:

$$P\left(X_{t_n} = j \,|X_{t_{n-1}} = i_{n-1},\, X_{t_{n-2}} = i_{n-2}, \ldots\right) = P\left(X_{t_n} = j \,|X_{t_{n-1}} = i_{n-1}\right) \tag{9.1}$$

The state describing process is memoryless to the visited states in the past and to the time spent in each state. Overall, the process of task scheduling can be considered as an MDP in which various types of RL-based control scheduling can be applied to the system. In this paper, the issue of task scheduling in a cluster-based data grid using an adaptive learning-based scheduling is studied.

In the past decade, there has been a plenty of well-studied works on immediate task scheduling in grid systems. Several of the scheduling strategies were mainly intended for computation-intensive task scheduling in computa-

tional grids. In addition to primary immediate task scheduling algorithms such as Opportunistic Load Balancing (OLB), Minimum Completion Time (MCT), Minimum Execution Time (MET), Switching Algorithm (SA), and K-Percent Best (KPA) [6, 7, 8], several different scheduling strategies have been also proposed such as a hybrid (GA/TS) independent task scheduling [9] for computational grids, a coloured petri net model for independent task scheduling in computational grids [10], a probabilistic task scheduling algorithm based on a discrete time Markov chain [11], an independent task scheduling based on Imperialist Competition Algorithm for grid systems [12] and a combined meta-heuristic (PSO with gravitational emulation local search) scheduling [13].

Regarding the ever-increasing needs for processing data-intensive tasks in scientific communities, several types of data-aware task scheduling strategies such as, HCS (Hierarchical Cluster Scheduling) [14], DIANA (data intensive and network aware scheduling) [15], RBHS (rank-based hybrid scheduling)[16], ASJS (adaptive scoring job scheduling) [17], CSS (Combined Scheduling Strategy) [18] and various scheduling strategies using meta-heuristic algorithms were proposed. Min-Min, Max-Min and Sufferage [7], RASA [19], FPLTF [20], and RRTS as the combination of Round Robin and Dynamic Time Slice (DTS) [21] are also some of the primary algorithms for batch mode scheduling of computation-intensive tasks. Regarding the dynamic workloads with a large complex of heterogeneous tasks, varying submission patterns and high heterogeneity of resources, the adaptive scheduling deserves to be used in many grid and cloud-based systems.

Reinforcement learning, as an important class of learning automata is a sort of learning based on iterative interaction with environment and analysis of the received reward signal. The learning is based on a type of trial-and-error search and delayed reward. RL is quite different from supervised learning, the most common learning in machine learning. Supervised learning is based on learning from training examples provided by an expert supervisor. But in an interactive environment, it is often impossible to have sufficient examples of desired behaviours covering all the state space. In these situations, it is highly beneficial if the learner is able to learn from its experiences. It is the exact benefit which is gained from reinforcement learning [22].

In general, reinforcement learning can be considered as an effective way of solving many types of optimal control problems, particularly the MDP ones. So, many of the optimal control solving methods are considered as reinforce-

ment learning solutions. However, many of them require partially complete knowledge of the environment. Three primary classes of reinforcement learning are Dynamic Programming (DP), Monte Carlo methods, and Temporal-Difference (TD) learning. DP methods need an accurate model of the environment and are computation-intensive. Monte Carlo solutions do not require complete knowledge of the environment and are simpler to be applied, but are not effective for step-by-step learning. The TD methods are free of model and well suited for incremental learning. It takes advantages of DP and Monte Carlo methods and learns directly from experiences without need for model [22]. Q-learning is one of the well-known algorithms in the category of reinforcement learning. It is an off-policy TD algorithm with early convergence. The efficiency of reinforcement learning has been shown in many dynamic application environments such as traffic control systems [23, 24], wireless sensor networks [25] and distributed control domains [26].

In this paper, a two-phase scheduling acting based on data awareness and using Q-learning algorithm was proposed for data-intensive task scheduling in a cluster-based data grid. In this study, a hierarchical multi-agent system consisting of two levels of broker agents was applied to the task scheduling in the cluster-based data grid. There is a global broker at the first level of the system, which makes decisions based on the data communication cost to select a suitable cluster. Then, at the second level the local brokers use a learning-based strategy to select the proper processing node.

Most of the previous studies used different parameters of data access cost, queue length or different combinations of them as primary optimization strategies for task scheduling in data grids. In this study, along with data communication cost, a Q-learning-based method has been used to improve the scheduling adaptation to dynamic changes and to high variability of submitted tasks. Exploiting an adaptive control method showed better performance than other common scheduling strategies for dynamic workloads with different task submission patterns.

This paper is organized as follows: In Section 9.2 a further overview of related works is presented. Section 9.3 discusses the learning concepts used in the proposed scheduling and presents the proposed two-phase learning-based scheduling for cluster-based data grids. Section 9.4 describes the simulation environment, evaluation scenarios and experimental results. Section 9.5 discusses performance evaluation of the proposed scheduling in comparison with

other task schedulings. Section 9.6 concludes the paper and presents some directions for further study.

9.2 Related Work

There are a number of related works for using RL-based methods in task scheduling in grids. In [27] a simple reinforcement learning was used for resource selection in a grid-like environment. In the proposed method, the learner keeps a score indicating the efficiency of the resource for each possible resource selection action. For scheduling a new submitted task, it selects the resource with the maximum score. Then, it receives a reinforcement signal and calculates a reward signal for the resource that has been selected. The simple proposed learning-based selection was applied to a distributed resource allocation in grid systems, but there was no explicit interaction between learners. The agents just learnt from expected response time of jobs as a reinforcement signal.

In [28] and [29] a dynamic resource selection called DRA-FRL was presented, which used RL in conjunction with a fuzzy rule base. A new RL-based method, Actor Critic Fuzzy Reinforcement Learning (ACFRL-2), was proposed to extend the application of RL to domains with large state-action space like dynamic resource allocation in grids or computer networks. Using RL in dynamic resource allocation is difficult, because the size of state space will increase dramatically with the number of resource types. In [30] a multi-agent reinforcement learning method called Ordinal Sharing Learning (OSL) was proposed to realize a learning-based coordination between agents with the aim of load balancing in large scale grids. The proposed learning was based on using an ordinal information sharing system with limited communication. In OSL, the agents make decisions based on shared utility tables.

In [31] a table-based RL called Sarsa was used for resource allocation in autonomic systems for a simple scenario where the state space is small. In [32] a multi-agent learning called Fair Action Learning (FAL) was proposed for online resource selection in a distributed sequential resource allocation problem (DSRAP). DSRAP refers to a resource allocation problem in a cluster-based network. FAL is a policy gradient ascent algorithm to learn the local decision policy. In [33] a decision-making framework of agents was proposed. It consists of two learning problems: local resource allocation and task routing

problem (choosing a neighbor to forward a task). In [33] a gradient ascent learning called Weighted Policy Learner (WPL) was proposed for a distributed task allocation in various applications like grids and web services.

In [34] an RL-based resource allocation combined with Artificial Neural Network (ANN) was proposed. This RL-based algorithm uses an ANN component to estimate the long-term reward through various iterations. In [35] an RL-based task scheduling in grid called Centralized Learning Distributed Scheduling (CLDS) was presented. It is a multi-agent scheduling consisting of one learner agent and several scheduler agents. In this scheduling, the scheduler agents submit their local rewards to the learner agent. The learner agent updates its global utility table and shares the updated utility table with the scheduler agents. The schedulers make decisions based on the updated utility table.

In this study, a two-phase adaptive task scheduling based on data-awareness and reinforcement learning for cluster-based data grids is proposed. It applies a hierarchical multi-agent system consisting of two levels of broker agents to task scheduling in a cluster-based data grid. The broker agent at the first level of the system makes decisions based on the data communication cost to select a suitable cluster. Then, at the second level, the local brokers use Q-learning to select the proper processing node. The proposed scheduling uses learning in conjunction with minimizing data communication cost.

9.3 Adaptive Scheduling Based on Reinforcement Learning

In large-scale grid systems, due to high variability and heterogeneity of submitted tasks and resource types, it is reasonable to have an adaptive scheduling to easily adapt to different changing conditions. To meet these requirements, an adaptive RL-based scheduling algorithm may be suited for this application environment. In the following, a brief discussion on the primary concepts of Q-learning as a model-free offline policy is presented, then the details of the proposed scheduling are described.

9.3.1 Q-Learning: A Model-Free Reinforcement Learning

Reinforcement learning addresses learning from the experiences. It states how an agent can learn an optimal policy to reach its goals by sensing the environment, taking possible actions and receiving consequent reward. In the standard RL, the agent continually monitors the environment. It observes the current state of the environment and selects an action based on its experiences to be applied to the environment. The selected action may affect/change the current state. The agent receives a reinforcement signal in a form of scalar reward as a result of the selected action and state transition.

In order to reach a learning goal, the agent is responsible to find a policy mapping states to actions in a way which maximizes the long-term cumulative reward. Generally, the reinforcement learning problem is formulated as a Markov Decision Process which is described by a tuple S, A, T, r, where S is the state space of the environment, A is the set of possible actions, T is the transition function which specifies the transition probability in state s by action a, and r is the reward function specifying the immediate reward after a transition to a new state from state s by taking action a.

In general, for learning the optimal policy, the goal of the learner is to maximize the total reward. If the model of environment is completely known, the optimal policy can be derived by DP method. But, in many problems, the model environment is unknown and there is no accurate knowledge of the environment. In these conditions, RL learns the optimal policy by trial-and-error experiences in the state space. Q-Learning is an RL algorithm which learns an action-value function, Q, estimating the long-term action-value. The learned action-value simply approximates the optimal action-value, independent of the policy being followed. All the Q-values are stored in a Q-table. At each step of learning, the Q-value is updated by

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha \left[r_{t+1} + \gamma \max_a Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)\right] \quad (9.2)$$

where $\alpha \in [0, 1]$ is the learning rate which specifies to what extent the agent learns new information and, $\gamma \in [0, 1]$ is the discount factor specifying the weight of future rewards in the action-value update and, r is the immediate reward [22]. Value 1 for the learning rate means that the agent considers only the latest information while value 0 causes the agent not to learn anything. A

value of 0 for discount factor shows that the agent only considers the current reward, while approaching value 1 will make the agent try for acquiring a long-term high reward. The normal procedural form of Q-Learning is shown in Fig. 9.1.

Q-learning will learn the optimal policy regardless of the policy that the agent follows for action selection. Since it learns the optimal policy by any policy which is followed, it is called off-policy TD learning [22]. In many straightforward implementations of Q-learning, the learner chooses the action with maximum Q-value at each step. To enhance the performance of learning, an exploration strategy is usually added to the algorithm. One of the standard ways is to introduce an additional value, epsilon, $0 < \varepsilon < 1$. A value, between 0 and 1, is generated randomly, if it is less than epsilon, a random action is chosen (exploration), otherwise the action with maximum Q-value is selected (exploitation). Randomly choosing the next action in conjunction with giving a higher probability to the actions that currently have higher Q-values may be another way to improve the exploration of the learning.

The convergence of Q-learning is also rather fast. It means that with a total number of transitions on the order of $Nlog(N)$, where N is the number of states, the agent can obtain the optimal policy [36].

Q-Learning:
1. Set the learning rate and discount factor parameters.
2. Initialize $Q(s,a), \forall s \in S, \forall a \in A$, arbitrarily.
3. Repeat for each episode:
    3.1. Initialize s
    3.2. Do while the terminal state hasn't been reached:
        3.2.1. Select one among all possible actions for the current state using the policy derived from Q (e.g. ε-greedy).
        3.2.2. Take this possible action, observe the next state & reward.
        3.2.3. Set the new Q-value by
        $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)]$
        $s \leftarrow s'$

Figure 9.1: Q-Learning algorithm

9.3.2 A Two-Phase Adaptive Scheduling Based on Data Awareness and Reinforcement Learning

In data grids, reducing data access cost plays an important role to decrease the tasks completion time and improve performance of scheduling. Data access cost is the required time to access data to process tasks. The model of computing data access cost in a cluster-based data grid is defined as follows:

If task T_i is scheduled on node S_j, the data communication cost for accessing required data of T_i from S_j is given by [14]

$$DCC_{S_j}^{T_i} = \sum_{\text{For all } Fl_K \ in \ R_i} |Fl_k|/B_{jK} \tag{9.3}$$

where R_i is the list of required replicas/data files to process the task, Fl_K is the k^{th} data replica in R_i, $|Fl_k|$ is the size of the replica, B_{jK} is the network bandwidth between node S and the source node of the k^{th} replica. According to the cluster-based topology of the data grid, the required data may be moved to node S_j from various nodes in other clusters or within the same cluster. The model of data communication cost can be defined as follows:

$$DCC_{S_j}^{T_i} = \ Inter\_C_{T_i,S_j} + Intra\_C_{T_i,S_j} \tag{9.4}$$

where $Inter\_C_{T_i,S_j}$ is the data communication cost for accessing the required replicas residing in different clusters from the origin cluster of T_i and $Intra\_C_{T_i,S_j}$ is the data communication cost for the required replicas which reside in the local cluster. The wide-area links between clusters are usually much slower than local networks within a cluster. Thus, the cost of data communication between clusters is more than the cost of data communication within a cluster. As a result, reducing the number of data communications between clusters for accessing distributed data is of great importance for data-intensive task scheduling in cluster-based grid systems.

The proposed algorithm is an immediate task scheduling based on a two-step decision process in which the first step is to select the cluster which contains the node(s) with the lowest data communication cost. The decisive factor of total data communication cost would be the cost of data communication among different clusters. At the next step, to improve the scheduling adaptation to the dynamic environment including dynamic workloads with varying task intervals, and high heterogeneity of submitted tasks and resources, an

adaptive reinforcement learning-based task assignment policy using Q-learning is used to select a proper node in the selected cluster with the minimum data communication cost.

The proposed adaptive scheduling applies a hierarchical multi-agent system to the scheduling process. It consists of one global broker agent at the first level and several local broker agents at the second level within the clusters. The global broker selects the proper cluster with minimum data communication cost, then the local broker inside the selected cluster exploits an adaptive task assignment policy based on Q-learning for selecting the proper processing node. Fig. 9.2 shows the general structure of the proposed scheduling. The local broker observes the current state, selects a proper node as a possible action, then receives the reward, and updates the Q-values.

The state of the environment is specified by a tuple $(n_{s1}, n_{s2}, ..., n_{sn})$ where n_{sn} represents the number of tasks (waiting and underprocessing) at node s_n of the local cluster. The possible actions are specified by a set of $\{a_{s1}, a_{s2}, ..., a_{sn}\}$ where a_{sn} represents the action of selecting node s_n. The reward function is defined by

$$reward = \frac{1}{Completiontime} \tag{9.5}$$

where, $Completiontime$ is the required time for the completion of a scheduled task. In the Q-learning based method, different action selection strategies can be used. In this study, two types of action selection methods will be used and investigated. One of them is a two-phase exploration-exploitation strategy based on the size of the submitted task set. The other one is ε-greedy algorithm. The completion time (response time) of task T_i on S_j, can be computed using the following equation:

$$CT_{S_j}^{T_i} = W_{T_i} + DAC_{S_j}^{T_i} + P_{T_i} \tag{9.6}$$

where W_{T_i} is the waiting time in the queue (queuing latency) to get the processing service, $DAC_{S_j}^{T_i}$ is the data access cost for task T_i, and P_{T_i} is the processing time of the data files. In the grid environment, in order to avoid data collision, the performance isolation for data transfer through connecting links should be guaranteed. Data access cost regarding waiting time for getting permission to transfer data is defined as follows:

$$DAC_{S_j}^{T_i} = DCC_{S_j}^{T_i} + Delay\_DT_{T_i} \tag{9.7}$$

where $Delay\_DT_{T_i}$ is the delay of data transfer in network links and $DCC_{S_j}^{T_i}$ is the data communication cost. Fig. 9.3 shows the steps of the proposed learning-based scheduling.

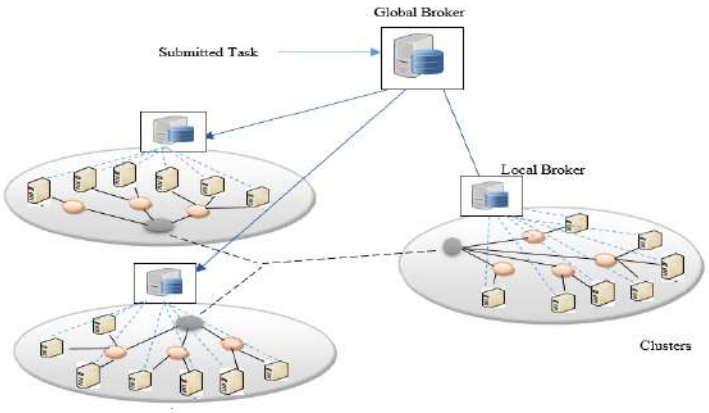


Figure 9.2: The structure of the proposed scheduling

Two-step adaptive task scheduling:
For each submitted task

A. Global broker
A.1 Calculates the inter-cluster data communication cost for executing the task in each cluster.
A.2 Selects the cluster with minimum data communication cost.

B. The local broker of the selected cluster
B.1 Observes the current state *(s)*.
B.2 Selects a host node (*action a*) using the action selection policy.
B.3 The submitted task is assigned to the selected node.
B.4 Receives a reward as a function of the completion time of the submitted task *(reward)*.
B.5 Updates the Q-value by

$$Q(s,a) \leftarrow Q(s,a) + \alpha[reward + \gamma \max_{a} Q(s',a) - Q(s,a)]$$

End For

Figure 9.3: Two-step learning-based task scheduling

9.4 Evaluation

In this study, OptorSim [3, 37], an open source data grid simulator, was used to simulate the proposed two-step learning-based scheduling with different action selection strategies and to perform the experimental evaluation. The purpose is to assess its performance compared with four baseline scheduling strategie sunder different workload patterns and analyze the effects of learning configuration parameters on the performance of the proposed scheduling. OptorSim, a java-based simulation tool developed under the European data grid project, supports simulation of data grids with different topologies, scheduling algorithms and various replication mechanisms.

In this study, the proposed adaptive scheduling has been implemented with three action selection strategies, and incorporated as anew scheduling into OptorSim. The performance of the scheduling is evaluated during three scenarios with three types of workloads, i.e., simple, random and CMS Data Challenge 2004 [3]. In each type of workload, several task sets with different number of tasks are submitted to the grid. The performance of the proposed scheduling algorithms is compared based on the makespan of the submitted task sets.

In the first step, through each scenario, the performance of the proposed scheduling using three action selection and three replication strategies under a specific type of workload was evaluated in terms of makespan. The first action selection strategy is a two-phase exploration-exploitation acting based on the size of the submitted task set. The other two action selection strategies use ε-greedy algorithm with $\varepsilon = 0.2$ and $\varepsilon = 0.5$ respectively.

At the next step, the sensitivity of the proposed scheduling algorithm to varying the learning parameters, i.e., learning and discount rates, is examined and the effects of the learning parameters on the scheduling performance are investigated.

The experiment environment including implementation details, the properties of simulation environment, performance and sensitivity analysis scenarios and simulation results will be described in the following Sections.

9.4.1 Simulation Environment

In OptorSim, data grid can be implemented with different topologies. The topology of the simulated cluster-based data grid and its structural properties

are given in Fig. 9.4. It consists of 3 clusters and 27 nodes in which 13 nodes have both computing and storage elements. The processing properties of the computing elements of all nodes are the same. The capacity of queue in computing elements was 200. A central node (node 17) is considered as a master storage node for storing master copies of all files. In the simulation environment, some nodes may have neither computing nor storage elements. They are used as network nodes. Connecting links between various nodes have different bandwidths. The network bandwidth between clusters is 500Mb/s and the band-width inside a cluster is 1000Mb/s for connections between nodes and first level switches, and is 2000Mb/s for connections between switches.

During the experiments, there were 300 initial files distributed randomly to the nodes of the grid. The size of a single file was 1GB. During the simulation, tasks were randomly selected from 30 task types based on the selection probability of each type. Each task type had the same probability of being selected. Each type of tasks requires different number of files.

In order to simulate the conditions of a real application environment and to provide high variability for submitted tasks, the distribution of task size was considered as Pareto, a heavy-tailed distribution. In this distribution, most of the task types had small size, i.e., required only a few number of files, and the remaining few types were of large size.

At the first step of evaluation, three scenarios using three types of workloads were arranged to evaluate the performance of the proposed learning-based scheduling in comparison to four baseline scheduling strategies. In each scenario, several sets of tasks with different numbers of tasks from 100 to 1200 were submitted to the data grid according to a specific submission pattern stated in the scenario.

Tasks were placed in the nodes according to the selected scheduling immediately after their arrival. Three replacement mechanisms such as Least Recently Used (LRU), Least Frequently Used (LFU) and Eco Model Optimizer (Binomial) were also used to manage the storage space for storing new replicas in the nodes during the task execution. LRU replicates the new required replica. It deletes the oldest file if there is no enough storage space for the new replica. LFU replicates the new replica while deleting the least frequently accessed file if there is no enough space. Eco Model Optimizer (Binomial) is a built-in replication optimizer in OptorSim which replicates the data file if deleting the least valuable file is economically beneficial according to binomial

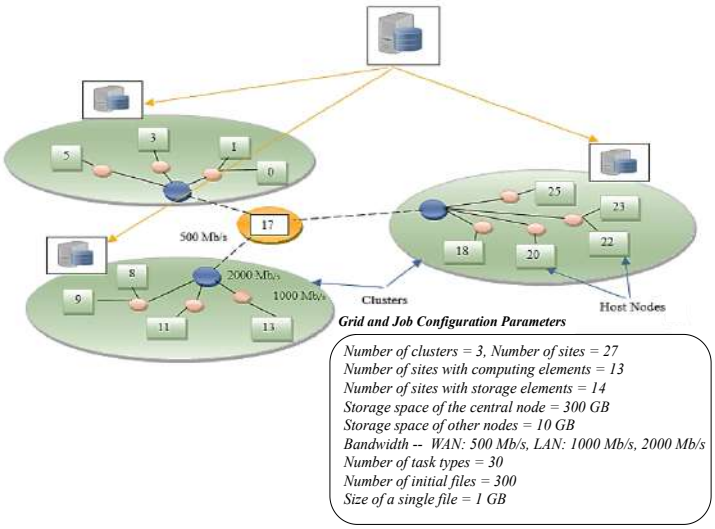


Figure 9.4: Topology of the simulated cluster-based data grid

prediction function.

At the second step of experimental evaluation, a sensitivity analysis scenario under CMS DC04 workload pattern regarding using LFU replication strategy is performed to examine how the learning parameters can affect the performance of the proposed learning-based scheduling.

9.4.2 Experimental Results

For the purpose of performance evaluation, we designed a number of experiments to examine how the proposed scheduling can work under different workloads and also how its performance can be affected by the values of learning parameters, i.e., learning and discount rates. In this study, the performance of the proposed learning-based scheduling was compared with four baseline scheduling algorithms including Queue Length (Shortest Queue), Access Cost, Queue Access Cost (QAC) and HCS during three performance analysis scenarios under different types of workloads. The mechanism of each baseline scheduling algorithm is as follows:

Queue Length schedules the task to the node with the shortest waiting

queue. In Access Cost, the task is assigned to the node with the lowest data communication cost for accessing the required data which is unavailable in the processing node. Queue Access Cost uses a combination of queue length and data communication cost to schedule tasks. HCS scheduling [14] uses a two-step decision making mechanism based on data transfer cost and the length of waiting queue, in order to decide the host processing node of the submitted task.

In each performance analysis scenario, the behaviours of scheduling algorithms are examined under a variety of task sets with different numbers of tasks that are submitted according to a submission pattern. In the experiments of performance analysis scenarios, three types of action selection strategy were used in the proposed learning-based scheduling and the performance of the proposed scheduling using different action selection strategies was compared with baseline scheduling algorithms. The first version of the proposed scheduling uses a two-phase exploration-exploitation strategy acting based on the number of tasks in the submitted task set. In this strategy, the random action selection, i.e., exploration, is used for the first half of the tasks in the task set. Afterwards, the exploitation strategy, which selects the action with the highest Q-value, is used for the second half of the tasks. The ε-greedy algorithm with $\varepsilon = 0.2$ and $\varepsilon = 0.5$ was used as the action selection strategy in the second and the third version of the proposed scheduling. The ε-greedy algorithm is one of the standard ways to make trade-off between exploration and exploitation in the action selection. The proposed scheduling in the performance analysis experiments is run with the configuration of learning rate $\alpha = 0.1$, and discount rate $\gamma = 0.5$.

9.4.3 Performance Analysis

Scenario 1. In this scenario, a simple workload pattern was used to evaluate performance of scheduling algorithms. In the simple workload, tasks are submitted at regular intervals until all tasks have been submitted. A fixed interval between tasks is set by parameter *delay* in the simulation environment. In this scenario, the tasks of each task set were submitted according to a simple submission pattern. The parameter of delay between tasks was set to 1000ms. The performance of the proposed learning-based scheduling with different action selection strategies was compared with baseline scheduling algorithms in terms of makespan of the task sets. Fig. 9.5 shows the behavior of evaluated

scheduling algorithms in terms of makespan under different task sets regarding using LRU, LFU, and Eco Model Optimizer replication strategies respectively.

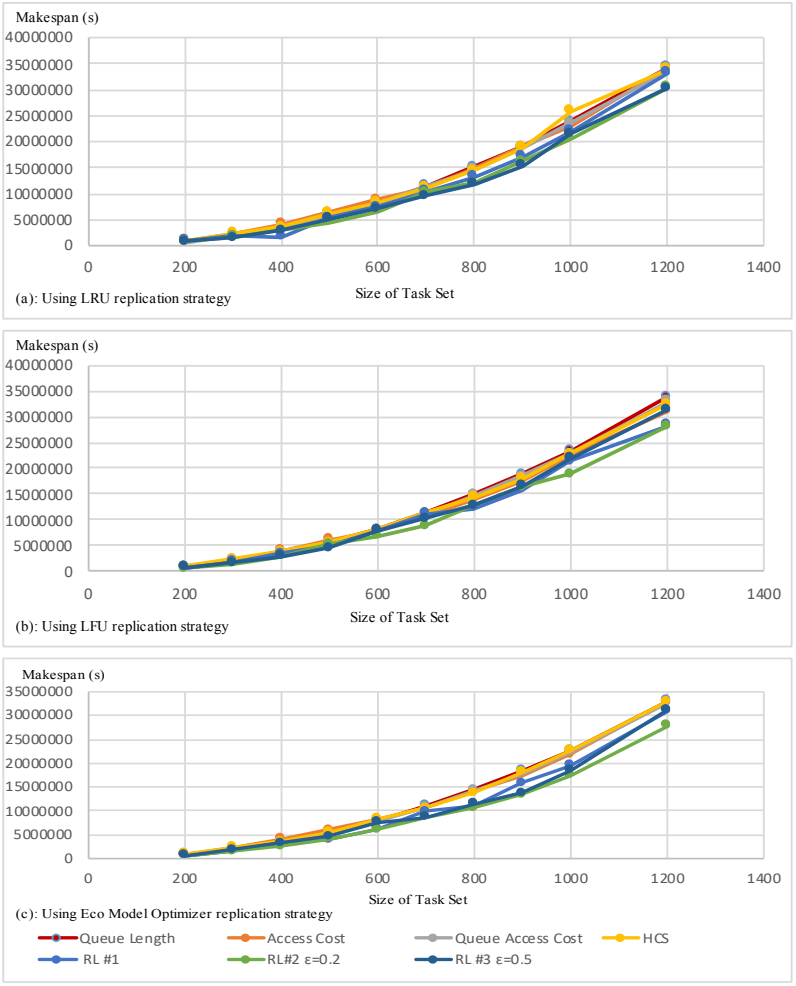


Figure 9.5: Scheduling algorithms' makespan regarding the simple workload

Scenario 2. In the second performance evaluation scenario, the tasks were submitted according to a random workload pattern. Random workload uses

uniformly random values for task intervals. The random intervals are between zero and twice the task delay parameter. In this scenario, the parameter of delay was set to 1000ms as well. Fig. 9.6 shows the makespan of the various task sets scheduled by different scheduling algorithms while LRU, LFU and Eco Model Optimizer were used as replication algorithms respectively.

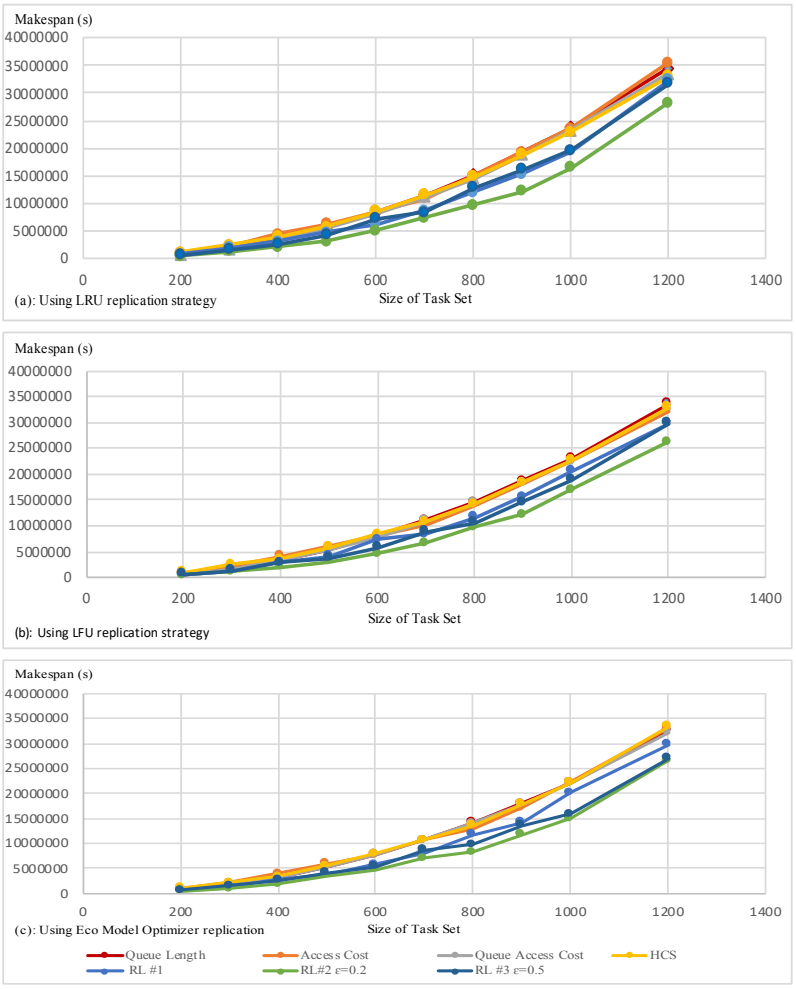


Figure 9.6: Scheduling algorithms' makespan regarding the random workload

Scenario 3. In the third performance evaluation scenario, CMSDC04 pattern was applied to task submission. This workload pattern uses a Gaussian distribution for submitted tasks. Fig. 9.7 presents the makespan of the various task sets scheduled by the algorithms with regard to the use of LRU, LFU, and Eco Model Optimizer as replication strategies.

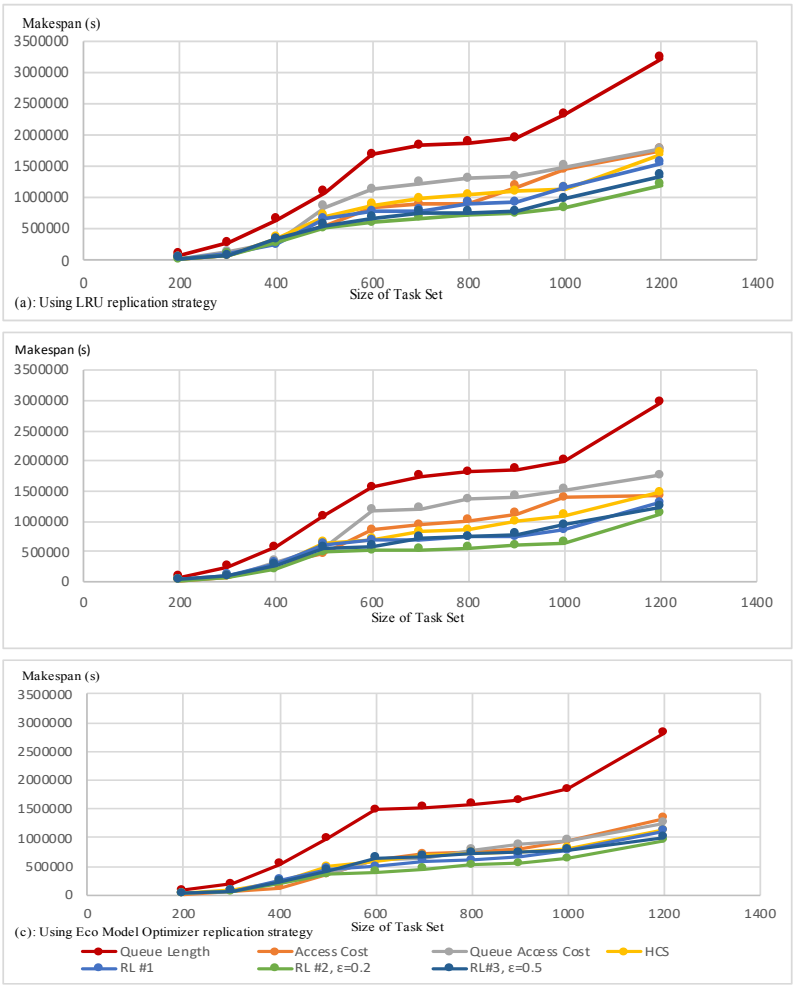


Figure 9.7: Scheduling algorithms' makespan regarding the CMS DC04 pattern

9.4.4 Sensitivity Analysis

The behavior of the learning-based scheduling can be affected by varying the learning rate (α) and discount rate (γ) as learning parameters. Two experiments were performed to analyze the effects of learning parameters on the performance of learning-based scheduling. The sensitivity analysis experiments are done with the second type of the proposed learning-based scheduling which uses ε-greedy with $\varepsilon = 0.2$, under CMS DC04 workload pattern and regarding LFU replication. Each experiment involves varying the values of one of the learning parameters, while keeping the other one constant. The first experiment characterizes the effects of learning rate (α) and the second one involves analyzing the effects of discount rate (γ) on the performance of learning-based scheduling. To examine the results of varying learning parameters, the discount rate (γ) was set to 0.5 in the first experiment and we set the learning rate (α) to 0.1 during the second experiment. Fig. 9.8 shows the impacts of varying learning parameters on makespan values of scheduling algorithms.

9.5 Discussion

The makespan plots of scheduling performance during the performance analysis experiments generally demonstrate lower values of makespan for submitted task sets which were allocated resources by learning-based scheduling than baseline scheduling algorithms. Consequently, the performance analysis results show that the learning-based algorithm adapts well to the workload pattern and status of the environment by scheduling tasks to the proper host nodes.

In the first scenario, simple workload was used for evaluating the performance of the proposed scheduling, using LRU, LFU, and Eco Model Optimizer replication strategies. In this scenario, according to Fig. 9.5, the learning-based scheduling algorithms mainly scheduled the task sets with lower makespan than other base-line algorithms. The learning-based algorithm which uses ε-greedy with $\varepsilon = 0.2$ gives the lowest makespan for task sets, specifically with increase in the size of task sets. The performance improvement caused by all versions of the learning-based scheduling were more considerable when Eco Model Optimizer has been used as replication strategy. On the other hand, under the simple workload pattern, the data grid will not face critical conditions like spikes in the number of submitted tasks.

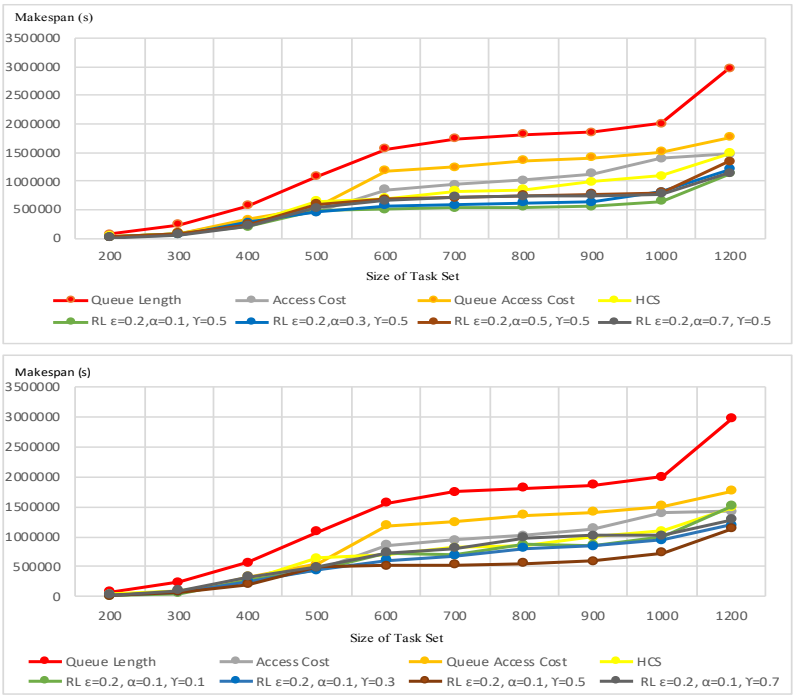


Figure 9.8: Learning parameters' impact on the learning-based scheduling performance

Consequently, during the simple workload, the scheduling performance of the algorithms are generally close to each other.

In the second scenario, a random submission pattern was used for performance evaluation of scheduling algorithms. Using random workload, all versions of the learning-based scheduling led to lower makespan for different submitted task sets than base-line scheduling algorithms. The learning-based scheduling using ε-greedy with $\varepsilon = 0.22$ led to the lowest makespan for submitted task sets among all versions of the proposed scheduling. The learning-based scheduling using ε-greedy with $\varepsilon = 0.2$ results in the most improvement regardless of the replication strategy used in the experiment. In general, the learning-based scheduling primarily acts adaptively to the changes and performs independently of replication strategies. During the random workload, the effectiveness of the learning-based scheduling ability to adapt to the status

of the grid and to learn the optimal policy is presented more than the simple workload. It led to a considerable performance improvement in terms of makespan measure.

In the third performance evaluation scenario, CMS DC04 pattern was used as task submission patter for the task sets. CMS DC04 uses a Gaussian distribution model for the tasks submitted over one day. During the experiments of the third performance evaluation scenario, the learning-based scheduling algorithms also worked better than other scheduling algorithms, regardless of the replication strategy. The amount of performance improvement mainly rises, specifically with increase in the size of task sets. With the CMS DC04 workload pattern, the learning-based scheduling which uses ε-greedy with $\varepsilon = 0.2$ also gave the most improvement among the other versions of the proposed scheduling.

Generally, in all the performance analysis experiments with the learning configuration of learning rate $\alpha = 0.1$ and discount rate $\gamma = 0.5$, the learning-based scheduling which uses ε-greedy with $\varepsilon = 0.2$ led to the most performance improvement. Using $\varepsilon = 0.2$ provides more exploitation than other action selection strategies and let the learner uses its learned experiences more.

With $\varepsilon = 0.2$ the local brokers act primarily based on the achieved experience stored in the Q-table, i.e., they select actions based on the Q-values. It implies that with a high probability, the learner selects an action with the highest utility value among the experienced actions, rather than a random action. Therefore, the contribution of the experience is more than simple exploration in the action selection. This lets the learners use their experience more than using random selection and exhibit a good play of learned policy for task scheduling. Almost, all experiments showed that the performance improvement of the learning-based scheduling using ε-greedy with $\varepsilon = 0.2$ is more considerable when there is an increase in the size of task sets; this is because the broker acts based on the learned policy which has been converged during more number of learning steps. In other words, the experience of the broker will be more accurate during the experiments with task sets of larger size.

Simulation results of the performance analysis experiments demonstrate that the learning-based scheduling can outperform other baseline scheduling strategies particularly under different workloads with changing features in dynamic environments. It can well adapt to the changing conditions given limited knowledge of the environment. It is also presented that using an action selec-

tion strategy with more tendency to exploitation leads to more performance improvement in the learning-based scheduling of different workloads.

In the sensitivity analysis experiments, the effects of varying learning parameters, i.e., learning and discount rates on the performance of learning-based scheduling are examined. The learning rate controls how fast the learner learns the policy, i.e., to what extent the new utility value affects the Q-value. The discount rate shows to what extent the learner concerns itself with maximizing the future rewards. Setting the learning rate to a high value causes the learner to consider only the new information and using a high value for the discount rate makes the learner take into account the future rewards strongly.

According to the simulation results of the sensitivity analysis in Fig. 9.8, the baseline learning configuration of learning rate $\alpha = 0.1$ and discount rate $\gamma = 0.5$ leads to the best performance in terms of makespan for the selected learning-based scheduling. Since the problem environment is stochastic, setting the learning rate to a low value like 0.1 and using a balance between impacts of immediate and future rewards by setting the discount rate to 0.5, provides the best performance for the learning-based scheduling.

9.6 Conclusion and Future Work

In grid systems, heterogeneity of submitted tasks and workload unpredictability are some of the important barriers to task scheduling in changing environments. Thus in order to improve the performance of task scheduling in dynamic environments, in this study, a two-step adaptive task scheduling based on data awareness and reinforcement learning was proposed for cluster-based data grids. The proposed adaptive scheduling consists of one global broker agent and several local broker agents inside the clusters. At the first step of the proposed scheduling, the global broker selects the cluster with minimum data communication cost. At the second step, in order to make the scheduling adaptive to changing features of the environment, a reinforcement learning-based task assignment policy based on Q-learning is used by the local brokers to select a proper node in the cluster selected at the first step. According to the experimental results, the proposed learning-based scheduling gives better performance, in comparison with other scheduling strategies. The performance improvement of the proposed learning-based scheduling is more considerable with increase in the number of tasks in varying workloads. Setting the learning

rate to a rather low value and putting a balance between the immediate and future rewards provide the best learning configuration for the learning-based scheduling in cluster-based data grids.

Applying cooperative multi-agent systems with cooperative learning to scheduling problems could be further directions for future study in the scope of applying machine learning techniques to control and management problems in grids and cloud-based environments.

Bibliography

[1] Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for grid scheduling problems. *Future generation computer systems*, 26(4):608–621, 2010.

[2] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of network and computer applications*, 23(3):187–200, 2000.

[3] DG Cameron, RC Schiaffino, J Ferguson, P Millar, C Nicholson, K Stockinger, and F Zini. Optorsim v2. 0 installation and user guide, november 2004.

[4] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[5] S Mehdi Vahidipour, Mohammad Reza Meybodi, and Mehdi Esnaashari. Learning automata-based adaptive petri net and its application to priority assignment in queuing systems with unknown parameters. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(10):1373–1384, 2015.

[6] Fatos Xhafa, Javier Carretero, Leonard Barolli, and Arjan Durresi. Immediate mode scheduling in grid systems. *International Journal of Web and Grid Services*, 3(2):219–236, 2007.

[7] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F Freund. Dynamic mapping of a class of inde-

pendent tasks onto heterogeneous computing systems. *Journal of parallel and distributed computing*, 59(2):107–131, 1999.

[8] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

[9] Fatos Xhafa, Juan A Gonzalez, Keshav P Dahal, and Ajith Abraham. A ga (ts) hybrid algorithm for scheduling in computational grids. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 285–292. Springer, 2009.

[10] Mohammad Abdollahi Azgomi and Reza Entezari-Maleki. Task scheduling modelling and reliability evaluation of grid services using coloured petri nets. *Future Generation Computer Systems*, 26(8):1141–1150, 2010.

[11] Reza Entezari-Maleki and Ali Movaghar. A probabilistic task scheduling method for grid environments. *Future Generation Computer Systems*, 28(3):513–524, 2012.

[12] Zahra Pooranian, Mohammad Shojafar, Bahman Javadi, and Ajith Abraham. Using imperialist competition algorithm for independent task scheduling in grid computing. *Journal of Intelligent & Fuzzy Systems*, 27(1):187–199, 2014.

[13] Zahra Pooranian, Mohammad Shojafar, Jemal H Abawajy, and Ajith Abraham. An efficient meta-heuristic algorithm for grid computing. *Journal of Combinatorial Optimization*, 30(3):413–434, 2015.

[14] Ruay-Shiung Chang, Jih-Sheng Chang, and Shin-Yi Lin. Job scheduling and data replication on data grids. *Future Generation Computer Systems*, 23(7):846–860, 2007.

[15] Richard McClatchey, Ashiq Anjum, Heinz Stockinger, Arshad Ali, Ian Willers, and Michael Thomas. Data intensive and network aware (diana) grid scheduling. *Journal of Grid computing*, 5(1):43–64, 2007.

[16] Mohsen Abdoli, Reza Entezari-Maleki, and Ali Movaghar. A rank-based hybrid algorithm for scheduling data-and computation-intensive jobs in grid environments. In *Intelligent Computing, Networking, and Informatics*, pages 785–796. Springer, 2014.

[17] Ruay-Shiung Chang, Chih-Yuan Lin, and Chun-Fu Lin. An adaptive scoring job scheduling algorithm for grid computing. *Information Sciences*, 207:79–89, 2012.

[18] Najme Mansouri, Gholam Hosein Dastghaibyfard, and Ehsan Mansouri. Combination of data replication and scheduling algorithm for improving data availability in data grids. *Journal of Network and Computer Applications*, 36(2):711–722, 2013.

[19] Parsa Saeed and R Entezari-Maleki. Rasa: A new task scheduling algorithm in grid environment. *World Applied Sciences Journal of Special Issue of Computer and IT*, pages 152–160, 2009.

[20] Daniel A Menascé, Debanjan Saha, SCD Porto, Virgilio AF Almeida, and Satish K Tripathi. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. *Journal of Parallel and Distributed Computing*, 28(1):1–18, 1995.

[21] Sanjaya Kumar Panda, Sourav Kumar Bhoi, and Pabitra Mohan Khilar. Rrts: A task scheduling algorithm to minimize makespan in grid environment. In *Proceedings of International Conference on Internet Computing and Information Communications*, pages 279–292. Springer, 2014.

[22] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[23] Mohamed A Khamis and Walid Gomaa. Adaptive multi-objective reinforcement learning with hybrid exploration for traffic signal control based on cooperative multi-agent framework. *Engineering Applications of Artificial Intelligence*, 29:134–151, 2014.

[24] Erwin Walraven, Matthijs TJ Spaan, and Bram Bakker. Traffic flow optimization: A reinforcement learning approach. *Engineering Applications of Artificial Intelligence*, 52:203–212, 2016.

[25] Hasan AA Al-Rawi, Ming Ann Ng, and Kok-Lim Alvin Yau. Application of reinforcement learning to routing in distributed wireless networks: a review. *Artificial Intelligence Review*, 43(3):381–416, 2015.

[26] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[27] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource allocation in the grid with learning agents. *Journal of Grid Computing*, 3(1-2):91–100, 2005.

[28] David Vengerov. Multi-agent learning and coordination algorithms for distributed dynamic resource allocation. 2004.

[29] David Vengerov. A reinforcement learning approach to dynamic resource allocation. *Engineering Applications of Artificial Intelligence*, 20(3):383–390, 2007.

[30] Jun Wu, Xin Xu, Pengcheng Zhang, and Chunming Liu. A novel multi-agent reinforcement learning approach for job scheduling in grid computing. *Future Generation Computer Systems*, 27(5):430–439, 2011.

[31] Gerald Tesauro, Rajarshi Das, William E Walsh, and Jeffrey O Kephart. Utility-function-driven resource allocation in autonomic systems. In *Second International Conference on Autonomic Computing (ICAC'05)*, pages 342–343. IEEE, 2005.

[32] Chongjie Zhang, Victor Lesser, and Prashant Shenoy. A multi-agent learning approach to online distributed resource allocation. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[33] Sherief Abdallah and Victor Lesser. Learning the task allocation game. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 850–857. ACM, 2006.

[34] Masnida Hussin, Nor Asilah Wati Abdul Hamid, and Khairul Azhar Kasmiran. Improving reliability in resource management through adaptive reinforcement learning for distributed systems. *Journal of parallel and distributed computing*, 75:93–100, 2015.

[35] Milad Moradi. A centralized reinforcement learning method for multi-agent job scheduling in grid. In *2016 6th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 171–176. IEEE, 2016.

[36] Michael J Kearns and Satinder P Singh. Finite-sample convergence rates for q-learning and indirect algorithms. In *Advances in neural information processing systems*, pages 996–1002, 1999.

[37] William H Bell, David G Cameron, A Paul Millar, Luigi Capozza, Kurt Stockinger, and Floriano Zini. Optorsim: A grid simulator for studying dynamic data replication strategies. *The International Journal of High Performance Computing Applications*, 17(4):403–416, 2003.