Poster: Performance Testing Driven by Reinforcement Learning

Mahshid Helali Moghadam*†, Mehrdad Saadatmand*, Markus Borg*, Markus Bohlin*, Björn Lisper†
* *RISE Research Institutes of Sweden* , Sweden
{mahshid.helali.moghadam, mehrdad.saadatmand, markus.borg, markus.bohlin}@ri.se
† *Mälardalen University* ,Västerås, Sweden
bjorn.lisper@mdh.se

***Abstract*—Performance testing remains a challenge, particularly for complex systems. Different application-, platform- and workload-based factors can influence the performance of software under test. Common approaches for generating platform- and workload-based test conditions are often based on system model or source code analysis, real usage modeling and use-case based design techniques. Nonetheless, creating a detailed performance model is often difficult, and also those artifacts might not be always available during the testing. On the other hand, test automation solutions such as automated test case generation can enable effort and cost reduction with the potential to improve the intended test criteria coverage. Furthermore, if the optimal way (policy) to generate test cases can be learnt by testing system, then the learnt policy can be reused in further testing situations such as testing variants, evolved versions of software, and different testing scenarios. This capability can lead to additional cost and computation time saving in the testing process. In this research, we present an autonomous performance testing framework which uses a model-free reinforcement learning augmented by fuzzy logic and self-adaptive strategies. It is able to learn the optimal policy to generate platform- and workload-based test conditions which result in meeting the intended testing objective without access to system model and source code. The use of fuzzy logic and self-adaptive strategy helps to tackle the issue of uncertainty and improve the accuracy and adaptivity of the proposed learning. Our evaluation experiments show that the proposed autonomous performance testing framework is able to generate the test conditions efficiently and in a way adaptive to varying testing situations.**

***Index Terms*—performance testing, stress testing, load testing, machine learning, reinforcement learning**

I. Research Challenges

Performance is a software quality characteristic which describes time and resource bound aspects of software behavior [1]. Performance testing is a family of techniques intended to accomplish the objectives of performance evaluation. The objectives are generally considered as measuring performance metrics (e.g. response time, throughput, and resource utilization) and finding performance issues such as specific functional problems emerging under certain execution conditions, e.g. heavy load, and violations of performance requirements. Performance testing techniques mainly involve executing software under test (SUT) under different normal and stress execution conditions. Finding performance issues such as performance breaking point, i.e., an execution condition under which the SUT becomes unresponsive or performance requirements are not satisfied anymore, is often challenging. The emergence of anomalies in the performance behavior of a SUT is generally a consequence of emerging performance bottlenecks. A bottleneck is a component at the system or platform level which limits the desired behavior of the software. The act of a bottleneck is due to many different situations such as saturation or contention that might occur in relation to the bottleneck component. Different application-, platform-, and workload-based conditions lead to emergence of bottlenecks and consequently affecting the performance behavior of the SUT. Generating performance test conditions such as platform- and workload-based test conditions to meet the performance testing objectives such as finding the performance breaking point is demanding and time-consuming in particular for complex systems.

II. Motivation and Background

Performance modelling and testing techniques are the common approaches to accomplish the objectives of performance evaluation. Model-driven approaches generally involve building a model of the performance behavior of the SUT using the associated modeling notations such as queuing theory, petri nets and Markov processes. It is mainly intended to measure the performance metrics and analyze the behavior of the SUT under different conditions. On the other hand, performance testing techniques execute the SUT under different test conditions and mainly rely on source code and system models to generate test cases. Common approaches for generating performance test cases could be categorized as follows:

Analysis of system model or source code. For example, analysis of system models such as UML [2], [3] and the control flow graph of the SUT [4] using evolutionary algorithms, analysis of performance models such as petri nets using constraint solving techniques [5], and also using data-flow together with symbolic execution [6] regarding the use of source code analysis.

Real usage modelling. It includes workload characterization using form-oriented models [7], [8], Extended Finite State Machines [9] and Markov chains [10] through monitoring the typical requests or user clustering based on the extraction of business-level features from the usage data [11].

Behavior-driven declarative methods. They involve using declarative Domain Specific Languages (DSL) along with

978-1-7281-5778-8/20/$31.00 ©2020 IEEE
DOI 10.1109/ICST46399.2020.00048

model-driven test execution frameworks, to specify the performance testing process and run the test cases [12], [13].

Machine learning-enabled methods. The use of machine learning in the analysis of data resulted from performance testing is more common. For example, using clustering techniques in analyzing metrics data for anomaly detection [14]. However, there are also a number of studies presenting the use of machine learning to generate/identify the performance test conditions, for example, using symbolic execution together with reinforcement learning (RL) to find the worst-case execution paths within SUT in a white-box fashion [15], using RL to find a sequence of input values resulting in performance degradation [16], and using a feedback-driven learning to identify the performance bottlenecks through rule extraction based on execution traces [17].

Although modeling provides a deep insight of the system behavior, drawing a well-detailed model is challenging and still there are some kinds of details that might be ignored during the modelling. On the other hand, the artifacts such as system models and source code which are used to generate the test cases might not be available during the testing. These issues are the motivations for using model-free machine learning techniques such as model-free RL [18] in which the smart tester can learn the optimal policy to accomplish the intended objective without access to the model or source code and also reuse the learnt policy in further testing situations.

III. Approach

How we address the challenge. The proposed solution is an autonomous performance testing framework involving two parts: a self-adaptive fuzzy RL-based (SaFReL) stress testing and an RL-driven load testing. They efficiently and adaptively generate performance test conditions, including platform-based conditions and test workload, which result in meeting the testing objective such as a target performance breaking point, without access to model or source code.

How the autonomous framework works. Both parts of the framework generally assume two phases of learning: initial and transfer learning. The smart tester agent learns the optimal policy to achieve the intended objective through the initial learning. It reuses the learnt policy in further performance testing scenarios while keeping the learning running in the long-term. Reusing the learnt policy is realized through replaying the learnt policy in certain situations such as testing SUTs with similarity in the performance sensitivity and meeting similar testing objectives (e.g. different values of error rate).

How it learns. we use a well-known model-free RL algorithm [18], Q-learning, as the core learning algorithm in our framework. An RL-based smart agent basically learns through interaction with the environment, the SUT and execution platform in our case. The interaction involves sensing the state of the environment, taking an action to influence the environment towards meeting the objective and receiving a reward signal indicating the effectiveness of the action applied. The agent's mission during the learning is maximizing the expected cumulative reward over the time, and the objective of the problem is formulated based on this mission. The agent uses an action selection strategy for choosing the actions, e.g. ϵ-greedy is used as the core strategy in our framework

A. RL-driven Stress Testing

The first part of the framework is an RL-driven autonomous stress tester which generates the stress platform-based test conditions for different SUTs efficiently and adaptively. The principles of RL in this part are formulated as follows:
- State: We use CPU, memory and disk utilization, and SUT response time as quality measures to model the state space.
- Actions: We define actions as a set of operations which modify the platform-based factors affecting the performance, i.e. available resource capacity.
- Reward signal: We propose a utility function as a weighted linear combination of two functions indicating the deviation of response time from the requirement and the amount of resource usage respectively.

We develop two types of the proposed RL-driven stress testing which are as follows:
Type I. We use the strategy of using multi-experience bases together with Q-learning to store the learnt policies for different types of SUTs separately based on the types of performance sensitivity. It leads to computation time saving and improvement in learning efficiency during the transfer learning. Fig. 1 shows the architecture of the RL-driven stress testing in type I [19].
Type II. It is a self-adaptive fuzzy RL (SaFReL) stress testing. As shown in Fig. 3, we use fuzzy logic for modeling the state space to tackle the issue of uncertainty in state classification and improve the accuracy of the learning. We also augment the agent with an adaptive action selection strategy which adjusts the parameters of the action selection based on the detected similarity between the performance sensitivity of SUTs. It leads to an improvement in the efficiency and adaptivity.

B. RL-driven Load Testing

The second part of the autonomous framework is an adaptive RL-driven load testing which generates an efficient test workload resulting in meeting the objective of the testing which can be, for example, reaching an intended error rate threshold. Then, the steps of RL are formulated as follows:
- State: Average response time and error rate resulted from submitting the test workload to the SUT are the quality measures used to model the state space of the environment.
- Actions: A set of operations adjusting the workload in terms of changing the load of the involved transactions. The involved transactions are application-specific and the requests involved in each transaction are extracted using a load recording tool like Apache Jmeter.
- Reward: We derive a function indicating the amount of resulted changes in response time and the maximum error rate which was met after applying an action, i.e. running a modified workload. It shows how effective the applied action was to reach the testing objective, e.g. intended error rate threshold. Fig. 2 shows the architecture of the RL-driven load testing

and how it uses Apache Jmeter as an actuator to execute the workload.

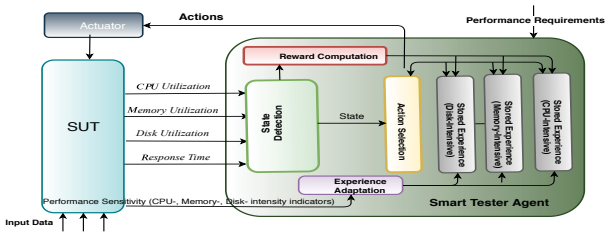

Fig. 1. RL-driven stress testing framework, Type I [19]

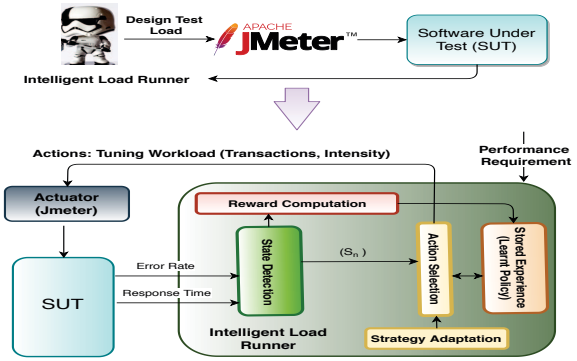

Fig. 2. RL-driven load testing

C. Learning Phases

Algorithms 1 and 2 present the procedures of learning phases in the proposed autonomous testing framework.

IV. Results and Conclusion

We evaluate the performance of the autonomous framework in terms of effectiveness, efficiency and adaptivity to accomplish the intended objectives, and also examine the behavioral sensitivity of the framework to the learning parameters. We conduct the experimental evaluation of the smart stress testing, SaFReL, on different instances of 12 well-known programs such as Build-apache, n-queens, dcraw, etc. which are characterized with different initial amounts of granted resources and response time requirements. We use Apache Jmeter and an online shop for experimenting on the smart load testing. An excerpt of our achievements is as follows:

The proposed RL-driven performance testing framework is able to accomplish the intended objectives efficiently and adaptively in different testing situations without access to system model or source code. For example, Table I shows that the RL-driven load testing meets an intended error rate threshold (testing objective) with a smaller and more accurate workload (less number of generated users) than a typical load testing procedure, which means lower cost and time in the testing. The two-phase learning together with adaptive action selection strategy also makes the agent able to reuse the learnt policy wherever it is useful and do exploration wherever it is required, which results in further cost savings (See Fig. 4).

Algorithm 1 Adaptive Reinforcement Learning-Driven 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 $\epsilon = \upsilon\ , 0 < \upsilon < 1$;
Initial Learning:
repeat
  (Fuzzy) Q-Learning (with initial action selection strategy, e.g. ϵ-greedy, initialized ϵ)
until *initial convergence*;
Transfer Learning:
while *true* **do**
  if *it acts as smart stress tester* **then**
    - Observe a new SUT instance;
    - Measure the performance sensitivity similarity between the new SUT and previously observed ones;
    - Apply self-adaptive strategy adaptation, i.e., adjust the degree of exploration and exploitation (e.g. tuning the parameter ϵ in ϵ-greedy);
  else
    `/* acts as smart load runner */`
    - Adapt the action selection strategy to transfer learning phase, i.e. tune parameter ϵ in ϵ-greedy;
  end
  Q-Learning with adapted strategy (e.g., new value of ϵ);
  end

Algorithm 2 Q-Learning

repeat
  1. Detect the (fuzzy) state (S_n) of the SUT;
  2. Select an action according to the action selection strategy, e.g. 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 in ϵ-greedy;
  3. Take the selected action:
  if *it acts as smart stress tester* **then**
    Modify the available resource capacity, re-run the SUT
  else
    `/* acts as smart load runner */`
    Adjust the workload and run the modified workload on the SUT
  end
  4. Detect the new (fuzzy) state (S_{n+1}) of the SUT;
  5. Compute the reward, R_{n+1};
  6. Update the q-value associated to the pair of previous state and taken action
  $Q(s_n, a_n) = \mu_n^s[(1-\alpha)Q(s_n, a_n) + \alpha(r_{n+1} + \gamma \max_{a'} Q(s_{n+1}, a'))]$
  μ_n^s, the membership degree of the fuzzy state, is set to 1 when the fuzzy classification is not used
until *meeting the stopping criteria (i.e. reaching the intended objective of the testing)*;

Fig. 4 shows less required computation time, in terms of learning trials, in transfer learning when the agent performs

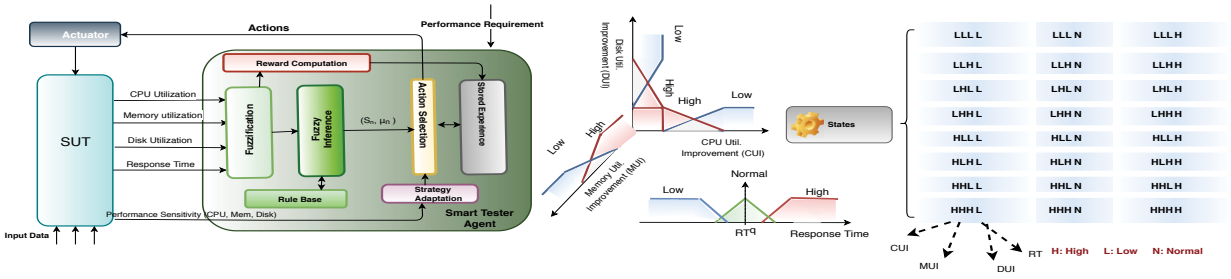

Fig. 3. Self-adaptive Fuzzy RL-driven stress testing, SaFReL (Type II) and associated fuzzy state modelling

TABLE I
EFFICIENCY OF RL-DRIVEN LOAD TESTING

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

on a homogeneous set of SUTs with respect to performance sensitivity. This capability of the smart agent leads to productivity benefits in terms of reduced computation time and cost for achieving the objectives of performance testing. Regarding applicability, the proposed approach could be beneficial to performance testing of software variants, evolving software in continuous Integration/Delivery process and regression performance testing.

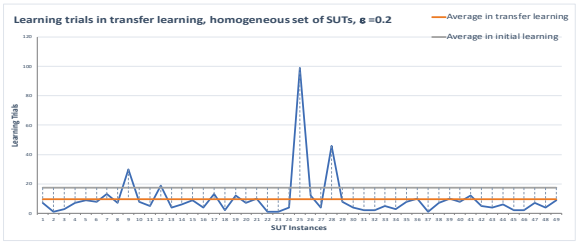

Fig. 4. Efficiency of SaFReL on a homogeneous set of SUTs

REFERENCES

[1] 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.

[2] V. Garousi, "A genetic algorithm-based stress test requirements generator tool and its empirical evaluation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 778–797, 2010.

[3] M. B. da Silveira, E. d. M. Rodrigues, A. F. Zorzo, L. T. Costa, H. V. Vieira, and F. M. de Oliveira, "Generation of scripts for performance testing based on uml models." in *SEKE*, 2011, pp. 258–263.

[4] Y. Gu and Y. Ge, "Search-based performance testing of applications with composite services," in *2009 International Conference on Web Information Systems and Mining*. IEEE, 2009, pp. 320–324.

[5] J. Zhang and S. C. Cheung, "Automated test case generation for the stress testing of multimedia systems," *Software: Practice and Experience*, vol. 32, no. 15, pp. 1411–1435, 2002.

[6] P. Zhang, S. Elbaum, and M. B. Dwyer, "Automatic generation of load tests," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 43–52.

[7] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, and G. Weber, "Realistic load testing of web applications," in *Conference on Software Maintenance and Reengineering (CSMR'06)*. IEEE, 2006, pp. 11–pp.

[8] C. Lutteroth and G. Weber, "Modeling a realistic workload for performance testing," in *2008 12th International IEEE Enterprise Distributed Object Computing Conference*. IEEE, 2008, pp. 149–158.

[9] M. Shams, D. Krishnamurthy, and B. Far, "A model-based approach for testing the performance of web applications," in *Proceedings of the 3rd international workshop on Software quality assurance*. ACM, 2006, pp. 54–61.

[10] C. Vögele, A. van Hoorn, E. Schulz, W. Hasselbring, and H. 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*, vol. 17, no. 2, pp. 443–477, 2018.

[11] G. Maddodi, S. Jansen, and R. 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*. ACM, 2018, pp. 200–210.

[12] V. Ferme and C. 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*. ACM, 2018, pp. 261–272.

[13] H. Schulz, D. Okanović, A. van Hoorn, V. Ferme, and C. Pautasso, "Behavior-driven load testing using contextual knowledge-approach and experiences," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ACM, 2019, pp. 265–272.

[14] H. Malik, H. Hemmati, and A. 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*. IEEE Press, 2013, pp. 1012–1021.

[15] J. Koo, C. Saumya, M. Kulkarni, and S. Bagchi, "Pyse: Automatic worst-case test generation by reinforcement learning," in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2019, pp. 136–147.

[16] T. Ahmad, A. Ashraf, D. Truscan, and I. Porres, "Exploratory performance testing using reinforcement learning," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019, pp. 156–163.

[17] M. Grechanik, C. Fu, and Q. Xie, "Automatically finding performance problems with feedback-directed learning software testing," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 156–166.

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[19] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "Machine learning to guide performance testing: An autonomous test framework," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2019, pp. 164–167.