Machine Learning-Assisted Performance Testing

Mahshid Helali Moghadam
RISE Research Institutes of Sweden, Mälardalen University
Västerås, Sweden
mahshid.helali.moghadam@ri.se

ABSTRACT

Automated testing activities like automated test case generation imply a reduction in human effort and cost, with the potential to impact the test coverage positively. If the optimal policy, i.e., the course of actions adopted, for performing the intended test activity could be learnt by the testing system, i.e., a smart tester agent, then the learnt policy could be reused in analogous situations which leads to even more efficiency in terms of required efforts. Performance testing under stress execution conditions, i.e., stress testing, which involves providing extreme test conditions to find the performance breaking points, remains a challenge, particularly for complex software systems. Some common approaches for generating stress test conditions are based on source code or system model analysis, or use-case based design approaches. However, source code or precise system models might not be easily available for testing. Moreover, drawing a precise performance model is often difficult, particularly for complex systems. In this research, I have used model-free reinforcement learning to build a self-adaptive autonomous stress testing framework which is able to learn the optimal policy for stress test case generation without having a model of the system under test. The conducted experimental analysis shows that the proposed smart framework is able to generate the stress test conditions for different software systems efficiently and adaptively without access to performance models.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; *Software performance*; • **Computing methodologies** → Machine learning

KEYWORDS

Performance testing, Stress testing, Test case generation, Reinforcement learning, Autonomous testing

ACM Reference format:

M. H. Moghadam. 2019. Machine Learning-Assisted Performance Testing. In *Proceedings of the 27th ACM ESEC/FSE '19, August 26–30, 2019, Tallinn*, Estonia. ACM, New York, NY, USA, 3 pages. https://doi.org/

ACM ISBN
https://doi.org/

1 RESEARCH PROBLEM

Performance describes the resource and time bound aspects of a software system behavior and is often described in terms of some sub-characteristics like time behavior, resource utilization and capacity [1]. Performance evaluation is generally conducted to I. measure the performance metrics, II. detect the functional problems appearing under certain performance-related execution conditions like heavy workload, III. detect non-functional problems, i.e., violations of non-functional requirements such as performance and robustness. Stress testing is often considered as a type of performance testing which implies applying extreme execution conditions to meet those objectives and verify the robustness of the system. Finding the performance breaking point of the software under test (SUT), at which the system functionality breaks, or the performance requirements are violated, is one of the main objectives in the stress testing activity. The emergence of anomalies in the performance behavior of a software system is resulted from performance bottlenecks. There are various application-, platform-, and workload-wise causes resulting in the emergence of performance bottlenecks. In stress testing, providing extreme (stress) test conditions involves changing (manipulating) the platform- and workload-wise factors affecting the performance. Generating stress test conditions to analyze the performance behavior under extreme conditions remains a challenge for complex software systems.

2 MOTIVATION AND BACKGROUND

Performance modeling and performance testing are common approaches for doing performance analysis. Performance modeling is often based on building a performance model of the system behavior and measuring the target performance metrics. It can be done using various modeling notations like Markov Processes, queueing networks, petri nets and simulation models [2, 3, 4, 5, 6, 7]. Performance testing is considered as a family of performance-related testing techniques intended for addressing the objectives of performance analysis. Performance, load and stress testing might considerably overlap in many areas. Nevertheless, the objectives of the performance-related testing methods could be summarized as follows:

I. Measuring the performance metrics under different execution conditions including various workload and resource configurations [8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

II. Detecting the functional problems appearing under certain execution conditions regarding workload and resource configurations [18, 19, 20, 21, 22, 23].

III. Detecting violations of non-functional requirements under expected and stress conditions [8, 13, 24, 25, 26, 27, 28, 29].

In general, using source code or system model analysis or use-case based design techniques are the common approaches for addressing the mentioned challenge. However, first, relying on the source code or system model like performance model might imply some limitations upon unavailability of these artifacts. Secondly, drawing a precise model of the performance behavior of a software system is challenging particularly for complex systems, while still many implementation and deployment details are often ignored. These are the motivations for using model-free learning-based techniques in which the optimal policy for addressing the problem can be learnt indirectly without having a model of the system and environment.

3 APPROACH

How it addresses the problem. The proposed solution involves a stress (robustness) test case generator to find the performance breaking point of SUTs. It is able to learn the optimal policy for generating stress test conditions to find the performance breaking point of the SUT without access to the performance model.

How it works. The proposed stress testing framework assumes two phases of learning, i.e., initial and transfer learning. The agent learns the optimal policy initially during the initial learning. During the transfer learning it replays the learnt policy in further analogous situations, i.e., upon observing SUTs with similar performance sensitivity, while keeping the learning running.

Learning Technique. Q-learning, i.e. a model-free reinforcement learning (RL) [30], is used as the core learning algorithm. In RL the agent senses the state of the system, which is the SUT in this case, continuously. Upon the state detection, it takes a possible action randomly or selects a high valued action. Then, it receives a reward signal indicating the effectiveness of the applied action. In the proposed framework the mentioned steps have been formulated as follows:

• State detection: The state of the system is identified based on the quality measurements of the SUT and execution environment, i.e., CPU, memory and disk utilization, and SUT response time.

• Actions: They are operations modifying (reducing) the factors affecting the performance, e.g. available resource capacity and characteristics of workload. ε-greedy was used as the core strategy for action selection.

• Reward signal: A utility function which is a weighted linear combination of two functions describing the response time deviation from the requirement and the resource usage respectively, was derived for the reward signal.

Profound Added Features.

I. At type I of this framework, Fig.1 shows its architecture, Q-learning augmented with experience adaptation through using multi-experience bases, was used. The smart tester agent uses separate experience bases for storing the learnt policy based on the type of performance sensitivity of the SUTs. It leads to the efficiency improvement of the agent in the transfer learning [31].

II. At type II, which is a self-adaptive fuzzy reinforcement learning-based (SaFReL) stress testing, as shown in Fig. 2, an action selection strategy adaption, was applied which acts as a meta-learning feature and is intended to improve the performance of the learning by applying adaptive changes to the action selection strategy based on detected differences between the performance sensitivity of observed SUTs. To address the issues related to the crisp categorization of the discrete state modelling, fuzzy classification was also used for state modeling (detection).

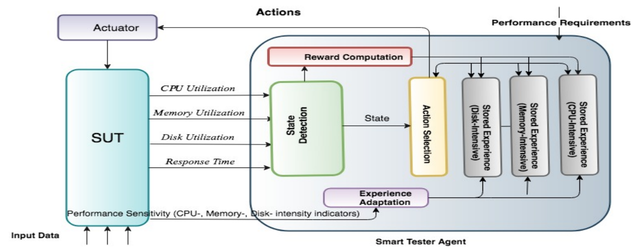
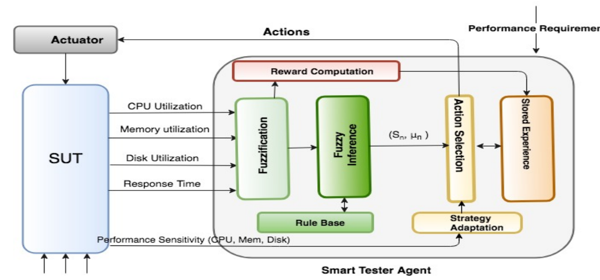


Figure 1: smart stress testing framework, type I



Algorithm SaFReL: Self-adaptive Fuzzy Reinforcement Learning-based Stress Testing

Required: S, A, α, γ; Initialize q-values, $Q(s,a) = 0 \; \forall s \in S, \forall a \in A$ and $\varepsilon = \upsilon,\ 0 < \upsilon < 1$

1. Observe the first SUT instance.
2. Repeat until initial convergence (*initial learning phase*):
   2.1. Fuzzy Q-Learning Episode with initial action selection strategy (e.g. ε-greedy, initialized ε)
3. Store the obtained experience
4. Start the transfer learning phase.
5. Repeat:
   5.1 Observe a new SUT instance
   5.2 Measure the similarity
   5.3 Apply strategy adaptation, i.e., adjust the degree of exploration and exploitation (e.g. tuning parameter ε in ε-greedy)
   5.4 Fuzzy Q-Learning Episode with adapted strategy (e.g., new ε)

Figure 2: Type II, SaFReL architecture and algorithm

4 CONCLUSION AND RESULTS

We have evaluated the efficacy of the proposed approach by simulating the performance behavior of 12 benchmark programs such as Build-apache, n-queens, dcraw, etc. Improved efficiency in terms of reduced effort, i.e., time and cost, for generating the test conditions while reducing dependency on source code and system models, is the achievement of the proposed learning-based stress testing. Regarding the applicability, software variants in software product lines and evolving software programs in CI/CD would be well-suited application areas for this approach. Extending the approach to support workload-wise factors in generating the stress test conditions is the current ongoing part of this research.

REFERENCES

[1] ISO/IEC 25010 - System and software quality models. Available at https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[2] D. Petriu, C. Shousha and A. Jalnapurkar. Architecture-based performance analysis applied to a telecommunication system. IEEE Transactions on Software Engineering, (11), 1049-1065, 2000.
[3] S. Bernardi, S. Donatelli and J. Merseguer. From UML sequence diagrams and state charts to analyzable petri net models. In Proceedings of the 3rd International Workshop on Software and Performance, pp. 35-45. ACM, 2002.
[4] E. D. Lazowska, J. Zahorjan, G. S. Graham and K. C. Sevcik. 1984. Quantitative system performance: computer system analysis using queueing network models. Prentice-Hall, Inc.
[5] V. Cortellessa, A. Di Marco and P. Inverardi. 2011. Model-based software performance analysis. Springer Science & Business Media.
[6] M. Harchol-Balter. 2013. Performance modeling and design of computer systems: queueing theory in action. Cambridge University Press.
[7] K. Kant & M. M. Srinivasan. 1992. Introduction to computer system performance evaluation. McGraw-Hill College
[8] J. Zhang and S. C. Cheung. Automated test case generation for the stress testing of multimedia systems. Softw. - Practice Experience, vol. 32, no. 15, pp. 1411–1435, 2002.
[9] D. A. Menasce. Load testing, benchmarking, and application performance management for the web. in Proc. Comput. Manag. Group Conf., pp. 271–281, 2002.
[10] J. Hill, D. Schmidt, J. Edmondson, and A. Gokhale. Tools for continuously evaluating distributed system qualities. IEEE Softw., vol. 27, no. 4, pp. 65–71. 2010.
[11] J. H. Hill. An architecture independent approach to emulating computation intensive workload for early integration testing of enterprise DRE systems. in Proc. Confederated Int. Conf., CoopIS, DOA, IS, and ODBASE 2009 On the Move to Meaningful Internet Syst., pp. 744–759. 2009.
[12] B. A. Pozin and I. V. Galakhov. Models in performance testing. Program. Comput. Softw., vol. 37, no. 1, pp. 15–25. 2011.
[13] M. Kalita and T. Bezboruah. Investigation on performance testing and evaluation of prewebd: A .net technique for implementing web application. IET Softw., vol. 5, no. 4, pp. 357–365. 2011.
[14] G. Casalezhnag, A. Kalbasi, D. Krishnamurthy, and J. Rolia. Automatic stress testing of multi-tier systems by dynamic bottleneck switch generation. in Proc. 10th ACM/IFIP/USENIX Int. Conf. Middleware, pp. 1–20. 2009.
[15] D. Krishnamurthy, J. Rolia, and S. Majumdar. Swat: A tool for stress testing session-based web applications. in Proc. Comput. Meas. Group Conf., pp. 639–649. 2003.
[16] D. S. Hoskins, C. J. Colbourn, and D. C. Montgomery. Software performance testing using covering arrays: Efficient screening designs with categorical factors. in Proc. 5th Int. Workshop Softw. Perform., pp. 131–136. 2005.
[17] M. Sopitkamol and D. A. Menasc_e. A method for evaluating the impact of software configuration parameters on e-commerce sites. in Proc. 5th Int. Workshop Softw. Perform., pp. 53–64, 2005.
[18] B. Dillenseger. Clif, a framework based on fractal for flexible, distributed load testing. Ann. Telecommun., vol. 64, pp. 101–120, 2009.
[19] C. Barna, M. Litoiu, and H. Ghanbari. Autonomic load-testing framework. in Proc. 8th ACM Int. Conf. Autonomic Comput., pp. 91–100. 2011.
[20] I. Schieferdecker, G. Din, and D. Apostolidis. Distributed functional and load tests for web services. Int. J. Softw. Tools for Technol. Transfer, vol. 7, pp. 351–360. 2005.
[21] P. Zhang, S. G. Elbaum, and M. B. Dwyer. Automatic generation of load tests. in Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng., pp. 43–52. 2011.
[22] D. Bainbridge, I. H. Witten, S. Boddie, and J. Thompson. Stress testing general purpose digital library software. in Research and Advanced Technology for Digital Libraries. Springer, pp. 203–214. 2009.
[23] L. C. Briand, Y. Labiche, and M. Shousha. Stress testing realtime systems with genetic algorithms. in Proc. Conf. Genetic Evolutionary Comput., pp. 1021–1028. 2005.
[24] A. Chakravarty, Stress testing an AI based web service: A case study. in Proc. 7th Int. Conf. Inf. Technol.: New Generations, pp. 1004–1008. 2010.
[25] A. Avritzer and E. J. Weyuker. The automatic generation of load test suites and the assessment of the resulting software. IEEE Trans. Softw. Eng., vol. 21, no. 9, pp. 705–716. 1995.
[26] S. Abu-Nimeh, S. Nair, and M. Marchetti. Avoiding denial of service via stress testing. in Proc. IEEE Int. Conf. Comput. Syst. Appl., pp. 300–307. 2006.
[27] V. Garousi. A genetic algorithm-based stress test requirements generator tool and its empirical evaluation. IEEE Trans. Softw. Eng., vol. 36, no. 6, pp. 778–797, 2010.
[28] V. Garousi. Empirical analysis of a genetic algorithm-based stress test technique. in Proc. 10th Annu. Conf. Genetic Evolutionary Comput., pp. 1743–1750. 2008.
[29] V. Garousi, L. C. Briand, and Y. Labiche. Traffic-aware stress testing of distributed real-time systems based on UML models using genetic algorithms. J. Syst. Softw., vol. 81, no. 2, pp. 161–185, 2008.
[30] R. S. Sutton, A. G Barto, et al.1998.Introduction to reinforcement learning. Vol. 135. MIT press Cambridge.
[31] M. Helali Moghadam, M. Saadatmand, M. Borg, M. Bohlin and B. Lisper. Machine Learning to Guide Performance Testing: An Autonomous Test Framework. ICST Workshop on Testing Extra-Functional Properties and Quality Characteristics of Software Systems ITEQS'19, 2019.