Title TBD

Rodrigo Araújo
University of British Columbia
Vancouver, British Columbia
rodarauj@cs.ubc.ca

Reid Holmes
University of British Columbia
Vancouver, British Columbia
rtholmes@cs.ubc.ca

ABSTRACT

Due to the advancement in distributed systems and the increasing industrial demands, software systems contain multiple components with complex interactions, e.g databases and their replication, caching components, proxies and load balancers, application instances and their complex configuration parameters. The engineers in a project must thinker with many configuration parameters that change the behavior and/or structure of the system, this can cause many problems that affect the quality of the service. In other words, dealing with high dimensionality is both cognitively demanding and risky for the project.

In this work we show the design and analysis of a pragmatic machine learning based tool that aims to assist the engineering of systems that can: 1) monitor themselves, 2) Forecast workloads and performance metrics and 3) Change themselves in run-time by self-configuring and adapting for a specific scenario. After the integration of this tool with a system, it should be able to answer the question: given that we have many configuration parameters, how can we change them in order to optimize a certain metric for a given predicted workload?

We show that it can decrease the risk of changing systems' configurations in run-time and decrease the engineering effort that otherwise would be spent manually optimizing parameters, usually following a trial-and-error approach.

CCS CONCEPTS

• **TBD** → **TBD**;

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference Format:

Rodrigo Araújo and Reid Holmes. 1997. Title TBD. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97)*, Rodrigo Araújo and Reid Holmes (Eds.). ACM, New York, NY, USA, Article 4, 3 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The industrial adoption of microservices has led to increasingly complex configuration schemes that are commonly fine-tuned by engineers manually. Ganek and Corbi (2003) discussed the need for autonomic computing to handle the complexity of managing software systems. They noted that managing complex systems has become too costly, prone to error, and labor-intensive, because people under such pressure make mistakes, increasing the potential of system outages with a concurrent impact on business [12]. This has driven many researchers to study self-adaptive systems over the years [4, 10, 11, 15, 21, 22]; however, the software industry still lacks practical tools to provide self-adaptation mechanisms to their systems. Thus, most of the configuring and tuning of the systems are performed manually, often in run-time, which is known to be a very time consuming and risky practice [2, 8, 12].

In this work we present an accessible tool to support the development of self-adaptive systems. One of our main goals is to provide such support requiring minimal effort from the engineers. In return our tool uses ideas from system observability, machine learning, and control theory to automatically asses the system's environment, predict the impact of changes that could potentially improve the system, and automatically make these changes.

Our approach consists of providing an API to collect relevant systems' metrics and configurations that represent the state of the system in relation to time. Then we map Service Level Objectives (SLOs) to some of these metrics, feed these into a machine learning component that is concurrently re-learning the model while analyzing and predicting the workload and the optimal configurations. As a result it provides adaptation plans that can be both 1) automatically executed, allowing the system to have self-adaptive capabilities, and 2) interpretable, allowing engineers to know the impact of a change in the configuration space before it is deployed.

In summary, our main contributions are:

- We provide a methodology to assist the development and evolution of self-adaptive systems, regardless of the presence of self-adaptability in the system's foundations. Such methodology is encapsulated in the tool described in this work.
- We show how to make the minimal necessary changes to the system, and how to model SLAs/SLOs and map them to the optimization objectives. These tasks being the development cost incurred by the engineers.
- We present a case study that shows how a software system's response time, throughput, and usage were improved by $A\%$, $B\%$, and $C\%$ respectively after integrating a self-adaptation mechanism.

The rest of this paper is structure as follows: in Section 2 we discuss some past research in the space of self-adaptive systems and provide fundamental background. In Section 3 we outline our approach, explaining the blend of ideas from different fields. In Section 4 we describe internal details and design decisions of our implementation. In Section 5 we present our case study followed by a discussion and future directions in Section 6. Finally, we conclude our findings with Section 7.

WOODSTOCK'97, July 1997, El Paso, Texas USA

ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

2 RELATED WORK

Between the years of 2007 and 2011, many techniques for forecasting workload and performance metrics have been realized [5, 6, 13, 14, 18]. These works did not focus on tools for applying machine learning to software systems, however, they provided techniques and methodologies for virtual machine allocation in data centers; They used techniques that overlap with our work, in special: time series analysis.

Herbst et al. contributed with a survey of state-of-the-art forecasting approaches based on time series analysis and a decision-based technique to select the best forecasting method based on user-specified objectives [15]. Although they did not focus on self-adaptability of systems in a finer-grained fashion, they did provide useful techniques to reliably forecast workload and performance, which is an important component in our tool – to enable self-adaptation in a system is to first understand the patterns in its context over time.

Andrew Pavlo et al in their work entitled *Self-Driving Database Management Systems* presented Peloton, a database system designed for autonomous operation. Similar to one of our goals, one of their main goals was to decrease the need for manually-performed operations, though they focused solely on their DBMS implementation. They achieved this by classifying the workload trends, collecting monitoring data, and forecasting resource utilization, then training a model based on this data to predict the best optimization plan. These ideas are important in our work, however, the key difference is that instead of directly embedding these ideas in a specific system – in this case a DBMS – and requiring the autonomous components to be tightly coupled, we are embedding a subset of these ideas in a tool that can be integrated in any chosen software system.

3 APPROACH

3.1 Control theory and self-adaptive systems

3.2 System's configuration as an optimization problem

3.3 Providing system adaptation with machine learning

3.4 Workload simulation

3.5 System instrumentation

3.6 Machine learning architecture

3.6.1 Features and models.

3.6.2 Online training.

3.6.3 Achieving self-adapation.

4 IMPLEMENTATION

5 EVALUATION

To guide and evaluate our work, four research questions are used:

- **RQ1:** Can self-adaptation by learned models lead to more stable, faster, and safer software systems, reducing the need to manually configure and tune?
- **RQ2:** How much instrumentation, SLO mapping, and configuration mapping is required to integrate the tool in a system?
- **RQ3:** How much performance overhead is incurred by using this tool?
- **RQ4:** What metrics and features have more impact on the tool's performance?

6 DISCUSSION AND FUTURE WORK

7 CONCLUSIONS

ACKNOWLEDGMENTS

REFERENCES

[1] [n. d.]. short term performance forecasting in enterprise systems. ([n. d.]). http://www.hpl.hp.com/techreports/2005/HPL-2005-50.pdf

[2] [n. d.]. Using probabilistic reasoning to automate software tuning. ([n. d.]). http://ftp.deas.harvard.edu/techreports/tr-08-04.pdf

[3] Virgilio AF Almeida. 2002. Capacity Planning for Web Services.. In *Performance*. Springer, 142–157. http://link.springer.com/content/pdf/10.1007/3-540-45798-4.pdf#page=151

[4] Joy Arulraj Haibin Lin Jiexi Lin Lin Ma Prashanth Menon Todd Mowry Matthew Perron Ian Quah Siddharth Santurkar Anthony Tomasic Skye Toor Dana Van Aken Ziqi Wang Yingjun Wu Ran Xian Tieying Zhang Andrew Pavlo, Gustavo Angulo. 2017. *Self-Driving Database Management Systems*.

[5] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. 2007. *Dynamic Placement of Virtual Machines for Managing SLA Violations*. 2007 10th IFIP/IEEE International Symposium on Integrated Network Management. 119–128 pages. https://doi.org/10.1109/inm.2007.374776

[6] Eddy Caron, Frédéric Desprez, and Adrian Muresan. 2011. Pattern Matching Based Forecast of Non-periodic Repetitive Behavior for Cloud Clients. *Journal of Grid Computing* 9, 1 (11 2011), 49–64. https://doi.org/10.1007/s10723-010-9178-4

[7] Ira Cohen, Jeffrey S. Chase, Moises Goldszmidt, Terence Kelly, and Julie Symons. 2004. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control.. In *OSDI*, Vol. 4. 16–16. http://sailhome.cs.queensu.ca/~corpaul/ciscxxx/papers/HPL-2004-183.pdf

[8] Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw (Eds.). 2013. *Software Engineering for Self-Adaptive Systems II*. Lecture Notes in Computer Science, Vol. 7475. Springer Berlin Heidelberg, Berlin, Heidelberg. http://link.springer.com/10.1007/978-3-642-35813-5 DOI: 10.1007/978-3-642-35813-5.

[9] Dirk Draheim, John Grundy, John Hosking, Christof Lutteroth, and Gerald Weber. 2006. Realistic load testing of web applications. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*. IEEE, 11–pp. http://ieeexplore.ieee.org/abstract/document/1602358/

[10] F. Faniyi, P. R. Lewis, R. Bahsoon, and X. Yao. 2014. Architecting Self-Aware Software Systems. In *2014 IEEE/IFIP Conference on Software Architecture*. 91–94. https://doi.org/10.1109/WICSA.2014.18

[11] Archana Sulochana Ganapathi. 2009. *Predicting and Optimizing System Utilization and Performance via Statistical Machine Learning*. Ph.D. Dissertation. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-181.html

[12] Alan G. Ganek and Thomas A. Corbi. 2003. The dawning of the autonomic computing era. *IBM systems Journal* 42, 1 (2003), 5–18.

[13] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. 2007. *Workload Analysis and Demand Prediction of Enterprise Data Center Applications*. 2007 IEEE 10th International Symposium on Workload Characterization. 171–180 pages. https://doi.org/10.1109/iiswc.2007.4362193

[14] Simon; Hedwig, Markus; Malkowski and Dirk Neumann. 2010. TOWARDS AUTONOMIC COST-AWARE ALLOCATION OF CLOUD RESOURCES. (2010). https://aisel.aisnet.org/icis2010_submissions/180

[15] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. 2014. Self-adaptive workload classification and forecasting for proactive resource provisioning. *Concurrency and Computation: Practice and Experience* 26, 12 (Aug. 2014), 2053–2078. https://doi.org/10.1002/cpe.3224

[16] P Horn. 2001. Autonomic Computing: IBM's Perspective on the State of Information Technology. 2007 (10 2001).

[17] Shijun Liu, Zeyu Di, Lei Wu, Li Pan, and Yuliang Shi. 2016. Probabilistic-based workload forecasting and service redeployment for multi-tenant services. *International Journal of High Performance Computing and Networking* 9, 1/2 (2016), 134. https://doi.org/10.1504/IJHPCN.2016.074666

[18] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. 2010. *Efficient resource provisioning in compute clouds via VM multiplexing*. 11–20 pages. https://doi.org/10.1145/1809049.1809052

[19] Theophano Mitsa. 2010. *Temporal data mining*. Chapman & Hall/CRC, Boca Raton, FL. OCLC: ocn474869534.

[20] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. ACM Press, 61–71. https://doi.org/10.1145/3106237.3106273

[21] Barry Porter, Matthew Grieves, Roberto Rodrigues Filho, and David Leslie. 2016. REX: A Development Platform and Online Learning Approach for Runtime Emergent Software Systems. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, GA, 333–348. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/porter

[22] M. Salehie and L. Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems (TAAS).

[23] Stepan Shevtsov, Mihaly Berekmeri, Danny Weyns, and Martina Maggio. 2017. Control-Theoretical Software Adaptation: A Systematic Literature Review. *IEEE Transactions on Software Engineering* (2017), 1–1. https://doi.org/10.1109/TSE.2017.2704579