initial configuration might be assumed similar, the most scaled configuration is not easily derived upfront due to the dynamics behind adaptations of single services and involved constraints. However, the approximated processing capabilities are used for designing scenarios for elasticity experiments in which both alternatives for autoscaling policies could be evaluated.

5 CONCLUSIONS AND FUTURE WORK

Existing reference applications for experimenting and research are not representative for data-intensive containerized cloud applications. They are serving more traditional use cases of human-centered request-reply communication without a continuous data processing pipeline and without using asynchronous messaging for the communication.

In attempt to fill this gap, in this work we present a reference use case coupled with the initial architecture design and with the engineering challenges for elasticity and resilience. To make the reference application suitable for research of elasticity and resilience mechanisms through increasing the predictability in performance, we have experimented with the CPU load generation tool ProtoCom. Variability of execution times differs for different cluster compilations due to unknown but speculated factors, however, ProtoCom reliably genereates CPU load in cloud environments. We observe a noticeable improvement after a major cloud maintenance and upgrade which illustrates the importance of repeating performance experiments in different times. We observe a moderate association between node occupancy and the relative variability. In addition, we sketch how the application could be scaled through two different autoscaling policies and investigate the scalability of the solution assuming a node-based approach.

In the future we plan to investigate further the factors that impact the high variability of the results. Moreover, we will perform additional experiments using container quotas and CPU throttling. Consolidating and publishing the reference application is another item for future work. In parallel we have started to construct a performance model of the application that will allow the evaluation of architecture alternatives. Moreover, we created initial simulation models to emulate the QoS-based shared CPU scheduling regime used in Kubernetes and alike. In the long run, we want to optimize and make the autoscaling and resilience of dynamic cloud applications more robust. On one side, by offering suitable modeling languages [13] to support architects for design-time analysis, and, on the other side by incorporating models of uncertain execution environments [21] into simulation-based prediction techniques.

ACKNOWLEDGMENTS

This paper was partly funded by the Federal Ministry of Education and Research under grant number 01IS18069A. For computational resources we acknowledge the bwCloud (https://www.bw-cloud.org), funded by the Ministry of Science, Research and Arts Baden-Württemberg (Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg).

REFERENCES

[1] ACME Air: Acme Air Sample and Benchmark . Online: https://github.com/acmeair/acmeair. [Online; 2022-01-25].

[2] Cluster Proportional Autoscaler. Online: https://github.com/kubernetes-sigs/cluster-proportional-autoscaler. [Online; 2022-01-25].

[3] Maintenance of all bwCloud regions starting 2022-03-01, subsequent reregistration necessary - FINISHED! |bw-cloud.org.

[4] Sockshop microservice demo application. Online: https://microservices-demo.github.io.

[5] Spring PetClinic. Online: https://github.com/spring-project:spring-petclinic. [Online; 2022-01-25].

[6] Kubernetes Horizontal pod autoscaling. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, 2021. [Online; 2021-12-09].

[7] Amazon EC2 Auto scaling. https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html, 2022. [Online; 2022-01-25].

[8] Steffen Becker, Tobias Dencker, and Jens Happe. Model-driven generation of performance prototypes. In Samuel Kounev, Ian Gorton, and Kai Sachs, editors, *Performance Evaluation: Metrics, Models and Benchmarks*, pages 79–98, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[9] André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the second international workshop on Software and performance - WOSP '00*. ACM Press, 2000.

[10] Bosch. Mobility Cloud. https://www.bosch-mobility-solutions.com/media/global/products-and-services/mobility-services/plcs/mobility-cloud/21-08-02_bosc_21028-06_mobilitycloud_onepager-rgb_en.pdf, 2021. [Online; 2021-12-16].

[11] Marco Di Natale, Haibo Zeng, Paolo Giusto, and Arkadeb Ghosal. *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice.* Springer Publishing Company, Incorporated, 2014.

[12] Sören Henning and Wilhelm Hasselbring. How to measure scalability of distributed stream processing engines? In *ICPE '21: ACM/SPEC International Conference on Performance Engineering, Virtual Event, France, April 19-21, 2021, Companion Volume*, pages 85–88. ACM, 2021.

[13] Floriment Klinaku, Mir Alireza Hakamian, and Steffen Becker. Architecture-based evaluation of scaling policies for cloud applications. In *IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2021, Washington, DC, USA, September 27 - Oct. 1, 2021*, pages 151–157. IEEE, 2021.

[14] Christoph Laaber, Joel Scheuner, and Philipp Leitner. Software microbenchmarking in the cloud. how bad is it really? *Empirical Softw. Engg.*, 24(4):2469–2508, aug 2019.

[15] Sebastian Lehrig, Richard Sanders, Gunnar Brataas, Mariano Cecowski, Simon Ivanšek, and Jure Polutnik. Cloudstore—towards scalability, elasticity, and efficiency benchmarking and analysis in cloud computing. *Future Generation Computer Systems*, 78:115–126, 2018.

[16] Sebastian Lehrig and Thomas Zolynski. Performance prototyping with protocom in a virtualised environment: A case study. *Proceedings to Palladio Days*, pages 17–18, 2011.

[17] Albert Lutz, Bernhard Schick, Henning Holzmann, Michael Kochem, Harald Meyer-Tuve, Olav Lange, Yiqin Mao, and Guido Tosolin. Simulation methods supporting homologation of electronic stability control in vehicle variants. *Vehicle System Dynamics*, 55(10):1432–1497, 2017.

[18] Seyed Hossein Nikounia and Siamak Mohammadi. Hypervisor and neighbors' noise: Performance degradation in virtualized environments. *IEEE Transactions on Services Computing*, 11(5):757–767, 2015.

[19] Chris Richardson. microservices.io deployment patterns. https://microservices.io/microservices/news/2015/03/15/deployment-patterns.html, 2021. [Online; 2022-01-25].

[20] Rami Rosen. Linux containers and the future cloud. *Linux J.*, 240(4):86–95, 2014.

[21] Max Scheerer, Martina Rapp, and Ralf Reussner. Design-time validation of runtime reconfiguration strategies: An environmental-driven approach. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 75–81, 2020.

[22] Piyush Shivam, Varun Marupadi, Jeffrey S. Chase, Thileepan Subramaniam, and Shivnath Babu. Cutting corners: Workbench automation for server benchmarking. In Rebecca Isaacs and Yuanyuan Zhou, editors, *2008 USENIX Annual Technical Conference, Boston, MA, USA, June 22-27, 2008. Proceedings*, pages 241–254. USENIX Association, 2008.

[23] Sandro Speth, Sarah Stieß, and Steffen Becker. A Saga Pattern Microservice Reference Architecture for an Elastic SLO Violation Analysis. In *Companion Proceedings of 19th IEEE International Conference on Software Architecture (ICSA-C 2022)*. IEEE, March 2022.

[24] Jóakim von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann, and Samuel Kounev. Teastore: A micro-service reference application for benchmarking, modeling and resource management research. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 223–236. IEEE, 2018.

[25] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Trans. Software Eng.*, 47(2):243–260, 2021.