CASPA: A Platform for Comparability of Architecture-based Software Performance Engineering Approaches

Thomas F. Düllmann[1], Robert Heinrich[2], André van Hoorn[1], Teerat Pitakrat[1], Jürgen Walter[3], Felix Willnecker[4]

[1]University of Stuttgart, Germany
[2]Karlsruhe Institute of Technology, Germany
[3]University of Würzburg, Germany
[4]fortiss GmbH, Germany

***Abstract*—Setting up an experimental evaluation for architecture-based Software Performance Engineering (SPE) approaches requires enormous efforts. This includes the selection and installation of representative applications, usage profiles, supporting tools, infrastructures, etc. Quantitative comparisons with related approaches are hardly possible due to limited repeatability of previous experiments by other researchers.**

This paper presents CASPA, a ready-to-use and extensible evaluation platform that already includes example applications and state-of-the-art SPE components, such as monitoring and model extraction. The platform explicitly provides interfaces to replace applications and components by custom(ized) components. The platform builds on state-of-the-art technologies such as container-based virtualization.

I. Introduction

As in other fields of software engineering, meaningful evaluations are required to investigate and quantify benefits of newly proposed approaches for Software Performance Engineering (SPE) [1]. For instance, when proposing a new approach for extracting architectural performance models from execution traces [2], the evaluation should address several aspects. These include the prediction accuracy of the obtained model and the comparison to related model extractors. Typically, an experimental evaluation is conducted in a lab setting, including *i)* the selection of an Application Under Test (AUT) for which a model is to be extracted, *ii)* a monitoring infrastructure that collects and provides performance data from the AUT, and *iii)* a load generator that produces synthetic requests to the AUT.

In many cases, these experiments have major threats to validity [3], e.g., because the selected AUTs and corresponding workloads are not representative. Moreover, the ability to repeat and reproduce the experiments is limited. It needs to be emphasized that setting up the experiments, requiring the aforementioned steps, is extremely time-consuming and error-prone. This is, for instance, caused by representative AUTs and load profiles not being available, the complexity of the infrastructure, as well as problems in the interoperability between the involved tools including the implementation of related approaches—if available at all.

To address these problems, we have developed an evaluation platform for SPE approaches, called CASPA.[1] It includes hooks for building blocks for typical SPE evaluations, comprising an AUT, a workload generator, a monitoring tool, and analysis approaches. Different implementations of the building blocks are already available and custom ones can be added easily. The platform comes with scripts allowing to setup the platform and start the experiments with ease. The platform is based on state-of-the-art technologies, such as container-based virtualization, which has been proposed as a promising approach for improving reproducibility in software engineering [4].

The remainder of this paper is structured as follows. Section II emphasizes the addressed problem by a representative example from our experience. Section III provides a conceptual overview of CASPA, technical details on the implementation, and the currently available components. Section IV describes a use case of the CASPA platform. Section V concludes the paper. The platform is publicly available online.[2]

II. Problem Statement

Various performance concerns need to be addressed during the life-cycle of a software system, e.g., whether it will satisfy the requirements w.r.t. timeliness in terms of response times. The SPE community has come up with various architecture-based SPE approaches using models and measurements to support the evaluation of these concerns in different stages of system development and operations [1], [5], [6].

As mentioned before, this paper addresses a common challenge in setting up lab experiments that aim to evaluate the quality of SPE approaches. In order to emphasize selected challenges, let us consider required steps based on an example, namely an approach for automatically extracting architecture-based performance models from monitoring data [7].

In addition to the implementation of the extractor to be evaluated, the experimentation setup requires the availability and integration of one or more case study application(s) (AUTs), workload generation [8], and an Application Performance Management (APM) tool [9]. In general, the efforts

[1]Comparability of Architecture-based Software Performance Engineering Approaches (CASPA)

[2]https://github.com/spec-rgdevops/CASPA-platform

To appear in Proceedings of IEEE International Conference on Software Architecture (ICSA 2017)

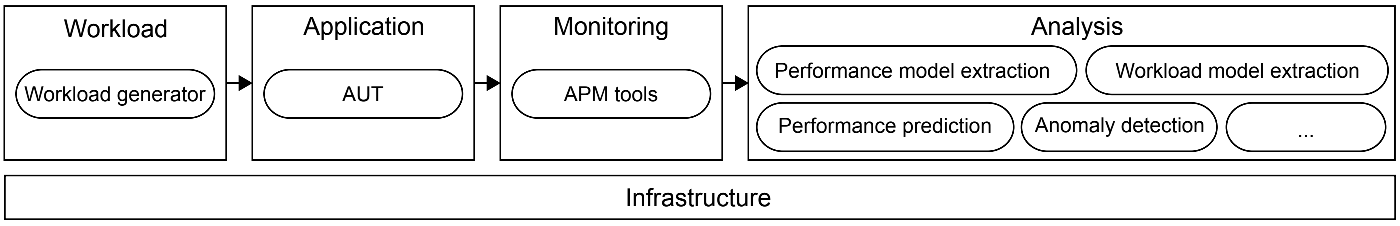

Figure 1. Overview of CASPA's architectural layers

for the application or evaluation of performance engineering approaches include:

- The AUT setup contains the preparation of the infrastructure (e.g., physical and/or virtual machines) and the installation of the application, which may require a separate deployment and integration with other services (e.g., a database management system).
- The workload generation involves the selection, configuration, and deployment of a load driver, as well as the specification of a representative usage profile.
- The setup of the analysis chain includes the individual deployment of analysis tools (the extractor in this case), as well as their integration. This is challenging, as there are different monitoring log formats, monitoring comprehensiveness levels (e.g., containment of resource utilization information). Moreover, through potentially different usage from what is intended, bugs or incomplete implementations may be revealed.
- The integration of the AUT and the monitoring/analysis chain poses additional challenges. Applications have to be extended to cope with unforeseen scenarios like incompatible formats or high amounts of measurements. This integration step often requires to identify and understand side effects (e.g., anomalies, garbage collection) that may require either to adapt or extend the initial analysis tooling or to interpret results accordingly.

Owing to the complex evaluation setup, there are several threats to validity that cannot be investigated due to very high setup efforts. External reproducibility would enable extended evaluation and reuse, and prevent reinventions. Hence, we require an easy set up and standardized interfaces between APM and SPE approaches.

III. CASPA Platform

This section describes CASPA's platform architecture, comprising the contained components and their interaction on a conceptual level, as well as selected implementations of the components, realized as independently deployable containers. The contributions in terms of tooling are *i)* the CASPA platform implementation and reusable and interoperable containers for different *ii)* AUTs, *iii)* APM tools, and *iv)* SPE tools representing the state of the art.

A. Architecture Overview

CASPA's high-level architecture is depicted in Figure 1. First, it is structured into the following five layers. The *application layer* comprises the AUT. The *workload layer* comprises the workload generator that imposes the AUT to synthetic workload based on a respective workload specification. The *monitoring layer* provides the APM infrastructure to collect performance measurements from the AUT and to provide this data for further analyses. The *analysis layer* comprises components for measurement-based and model-based performance evaluation, including model extraction. As depicted by the arrows in Figure 1, the analysis may provide feedback to the other layers. The *infrastructure layer* provides the execution environment for the other layers and components, e.g., in terms of the hardware, operating system, and (container-based) virtualization.

Table I. Current technologies and tools used by CASPA

Platform Layer	*Technologies & tools*
Workload	Faban, JMeter, Locust
Application	Netflix OSS RSS reader, CoCoME, SPECjEnterprise2010
Monitoring	Kieker, inspectIT
Analysis	
Model extraction:	iObserve, PMX, WESSBAS
Measurement-based analysis:	Kieker
Performance prediction:	Palladio, Descartes
Infrastructure	Docker, Kubernetes

Apart from the infrastructure layer, each layer is explicitly intended to be tailored based on the specific needs of the respective evaluation that uses the platform. Particularly, custom components can be added. The remainder of this section details selected technical components for the implementation of the layers, as summarized in Table I.

B. Infrastructure Layer

The platform uses container-based virtualization with Docker, managed by the container orchestrator Kubernetes.

Docker[3] is a scriptable lightweight virtual machine. Its ease of use and integration into build and deployment pipelines support the goals of our platform [4]. All components of our platform are based on the Docker infrastructure.

Kubernetes[4] is a container management system that provides mechanisms to manage Docker containers on distributed environments at large scale. It allows various operations such as deployment, scaling, load-balancing, service discovery, rolling-update, and self-healing. Setting up these features would require more effort in traditional setups.

[3]https://www.docker.com/
[4]https://kubernetes.io/

C. Application Layer

On the application layer, we currently provide the following AUT containers:

The *Netflix RSS Reader*[5] is a demonstration application created by Netflix to show how their open-source components interact. It provides a web page to users that allows them to view, add, and delete RSS feeds. The RSS reader application is composed of two microservices that can be scaled independently; the edge microservice provides a user interface, the middletier microservice stores feed URLs in the database and retrieves the feed contents.

The *Common Component Modeling Example (CoCoME)*[6] resembles a trading system of a supermarket chain. CoCoME implements processes for sales, ordering products, and inventory management. It is used as a platform for collaborative empirical research on information system evolution [10], [11].

The *SPECjEnterprise2010* benchmark is a Java Enterprise Edition (EE) application that was designed to benchmark application servers and their standardized Java EE interfaces. The application mimics a car order and manufacturing system.

D. Workload Layer

We use and adapt multiple workload drivers based on different base technologies. The load drivers are all packed in individual containers as part of our platform. In the current state, we use three open-source load testing tools, namely JMeter, Locust, and Faban. However, depending on the AUT, other drivers could be added to the platform.

JMeter[7] supports a large number of protocols such as HTTP, SOAP, FTP, JDBC. JMeter can run in a headless and in a graphical mode, and provides analysis modules to interpret the results of a stress or load test.

Locust[8] executes a workload profile written in Python. This allows high capability and flexibility in defining user behavior. Locust can run in two modes: standalone and distributed. In the distributed mode, multiple instances of slave nodes can be spawned and the master node delegates the load creation to them to create a distributed workload.

Faban[9] is a benchmarking suite designed to conduct reproducible benchmarks and load tests. Evaluating the performance and scalability characteristics of server-based systems is the core design principle of this tool. Faban is used, for instance, as a load test driver for the SPECjEnterprise2010 industry-standard benchmark.

E. Monitoring Layer

On the monitoring layer, the following APM tool containers are currently available:

Kieker [12] is an extensible open-source APM tool. Particularly, it allows to capture and analyze execution traces from distributed software systems. The AUT is instrumented and monitoring agents send their data to a central server container that makes the data available via a REST API to other components.

inspectIT[10] is an open-source solution for APM. As such, inspectIT allows for monitoring the health of a productive software application, get notified in cases of performance issues, and provides comprehensive means to analyze root causes of performance problems. Employing an instrumentation-based approach for data gathering inspectIT gives a transactional view into the internals of an application.

F. Analysis Layer—Model Extraction

For the analysis layer, we list selected approaches for extracting performance and workload models from APM data.

Performance Model Extractor (PMX) [7] is a framework for the extraction of architectural performance models generalizing over the target modeling language. Currently, PMX supports the extraction of Palladio Component Model (PCM) [13] and Descartes Modeling Language (DML) instances. The PMX approach enables an easy comparison of architectural performance modeling languages and access to different tool chains. The container reads APM monitoring data to be processed via a REST API.

WESSBAS [14] is an approach for extracting usage profiles from operational monitoring logs into instances of a domain-specific language, and transforming these instances into performance models (e.g., PCM) or load test scripts (e.g., JMeter).

iObserve [15] is a run-time modeling approach based on PCM that uses transformations to update architectural models by operational observations (particularly, APM data). iObserve considers changes in workload, component migration, and component (de-)replication, as well as (de-)allocation of resources in monitoring and model transformation.

Manual or automatic analysis of the generated models are, as of today, not part of our platform. However, this is a potential extension point for other researchers and practitioners. Architecture or deployment optimizations based on the extracted models are just a few of potential examples of complex SPE tasks that require a complete environment as proposed here [16], [17].

IV. Use Case

Our platform consists of a number of components that are interoperable and can thus be easily exchanged. This allows to assemble a customized setup for experimental evaluation of SPE approaches.

The infrastructure layer, comprising a Docker and Kubernetes installation, needs to be set up only once and can be used for different experiments. Kubernetes can be deployed on both physical or virtual environments, such as bare-metal machines or OpenStack.[11] There is also an easy way to deploy Kubernetes on a local machine which can be used as a platform for application development and testing.

[5]https://github.com/Netflix/recipes-rss

[6]http://www.cocome.org/

[7]http://jmeter.apache.org/

[8]http://locust.io/

[9]http://faban.org/

[10]http://www.inspectit.rocks/

[11]https://www.openstack.org/

Depending on the experiment goal, the set of components for each layer needs to be selected and deployed to Kubernetes. As an example, let us consider the setting mentioned in the problem statement, namely extraction of architectural models. The approach analyzes the monitoring data obtained from a running application. The components required for this experiment include a workload generator, an AUT, a monitoring framework, and an analysis tool. One possible configuration for this experiment is using Locust for workload generation, RSS reader as an AUT, Kieker for application performance monitoring, and PMX for model extraction. Such an experiment configuration can be specified in a single file which allows an easy deployment of all components.

If a new configuration is needed, the configuration file can be modified and re-deployed. For example, the RSS reader can be replaced by CoCoME or SPECjEnterprise. In the same way, the workload generator and monitoring tool can be replaced by other existing ones. In order to use components that have not been integrated into CASPA, they need to be provided as a Docker image along with a corresponding configuration file.

V. Conclusion

CASPA provides a ready-to-use and extensible platform for the quantitative evaluation of architecture-based SPE approaches. Particularly, we address the problem that, so far, setting up these experiments was extremely time-consuming and error-prone due to the complexity of the involved tools. As a consequence, experiments have threats to validity, e.g., setting up the experiment already with only a single AUT is costly in terms of effort.

We hope that our effort paves a way for efficiently setting up experimental evaluations that provide a higher degree of validity, repeatability, reproducibility, and comparability. Our goal is that other researchers use the platform for their evaluation purposes—including new AUTs and SPE/APM components, which are then made publicly available.

The presented work is still in progress. In the near future, in particular, we want to extend the set of included AUTs and SPE components, e.g., for detection, diagnosis, and prediction of performance problems, and auto-scaling, which are currently mainly limited to works from the research groups of the authors. Apart from new components, our goal is to extend the platform by concepts from DevOps and Continuous Software Engineering [18], [19], e.g., a continuous deployment pipeline. This allows to evaluate performance-related approaches for this challenging and promising environment [5].

VI. Acknowledgment

This work is conducted as a joint activity of several researchers as part of the Research Group of the Standard Performance Evaluation Corporation (SPEC RG, http://research.spec.org). This work is being supported by the German Research Foundation (DFG) in the Priority Programme "DFG-SPP 1593: Design For Future—Managed Software Evolution" (Declare: HO 5721/1-1 and KO 3445/15-1; iObserve: RE 1674/6-1), and the German Federal Ministry of Education and Research (diagnoseIT: 01IS15004).

References

[1] C. M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *Proc. Int. Conf. on Software Engineering (ICSE 2007), Workshop on the Future of Software Engineering, (FOSE 2007)*, 2007, pp. 171–187.

[2] D. Okanović, A. van Hoorn, C. Heger, A. Wert, and S. Siegl, "Towards performance tooling interoperability: An open format for representing execution traces," in *Proc. 13th European Workshop on Performance Engineering (EPEW '16)*. Springer, 2016, pp. 94–108.

[3] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell, *Experimentation in Software Engineering*. Springer, 2012.

[4] J. Cito and H. C. Gall, "Using Docker containers to improve reproducibility in software engineering research," in *Comp. 38th Int. Conf. on Software Engineering (ICSE '16)*. ACM, 2016, pp. 906–907.

[5] A. Brunnert, A. van Hoorn, F. Willnecker, A. Danciu, W. Hasselbring, C. Heger, N. Herbst, P. Jamshidi, R. Jung, J. von Kistowski, A. Koziolek, J. Kroß, S. Spinner, C. Vögele, J. Walter, and A. Wert, "Performance-oriented DevOps: A research agenda," SPEC Research Group — DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC), Tech. Rep. SPEC-RG-2015-01, Aug. 2015.

[6] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Perform. Eval.*, vol. 67, no. 8, pp. 634–658, 2010.

[7] J. Walter, C. Stier, H. Koziolek, and S. Kounev, "An expandable extraction framework for architectural performance models," in *Proc. 3rd Int. Workshop on Quality-Aware DevOps (QUDOS'17)*. ACM, April 2017.

[8] Z. M. Jiang and A. E. Hassan, "A survey on load testing of large-scale software systems," *IEEE Trans. Software Eng.*, vol. 41, no. 11, pp. 1091–1118, 2015.

[9] C. Heger, A. van Hoorn, D. Okanović, and M. Mann, "Application performance management: State of the art and challenges for the future," in *Proc. 8th ACM/SPEC Int. Conf. on Performance Engineering (ICPE '17)*. ACM, 2017.

[10] S. Herold *et al.*, "CoCoME – the common component modeling example," in *The Common Component Modeling Example*. Springer, 2008, pp. 16–53.

[11] R. Heinrich, S. Gärtner, T.-M. Hesse, T. Ruhroth, R. Reussner, K. Schneider, B. Paech, and J. Jürjens, "A platform for empirical research on information system evolution," in *Proc. 27th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE 2015)*, 2015, pp. 415–420.

[12] A. van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A framework for application performance monitoring and dynamic software analysis," in *Proc. 3rd ACM/SPEC International Conference on Performance Engineering (ICPE '12)*. ACM, Apr. 2012, pp. 247–248.

[13] R. H. Reussner *et al.*, *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, 2016.

[14] 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," *Journal on Software and System Modeling (SoSyM)*, 2016.

[15] R. Heinrich, "Architectural run-time models for performance and privacy analysis in dynamic cloud applications," *Perform. Eval. Rev.*, vol. 43, no. 4, pp. 13–22, 2016.

[16] A. Koziolek, H. Koziolek, and R. Reussner, "PerOpteryx: Automated application of tactics in multi-objective software architecture optimization," in *Proc. 7th Int. Conf. on the Quality of Software Architectures (QoSA 2011) and 2nd Int. Symp. on Architecting Critical Systems (ISARCS 2011)*. ACM, 2011, pp. 33–42.

[17] F. Willnecker and H. Krcmar, "Optimization of deployment topologies for distributed enterprise applications," in *Proc. 12th Int. ACM SIGSOFT Conf. on Quality of Software Architectures (QoSA 2016)*, 2016.

[18] L. J. Bass, I. M. Weber, and L. Zhu, *DevOps - A Software Architect's Perspective*. Addison-Wesley, 2015.

[19] J. Bosch, Ed., *Continuous Software Engineering*. Springer, 2014.