Demo Paper: Benchmarking Scalability of Cloud-Native Applications with Theodolite

Sören Henning, Wilhelm Hasselbring
Kiel University, Software Engineering Group
{soeren.henning,hasselbring}@email.uni-kiel.de

***Abstract*—Theodolite is a framework for benchmarking the scalability of cloud-native applications. It automates deployment and monitoring of a cloud-native application for different load intensities and provisioned cloud resources and assesses whether specified service level objectives (SLOs) are fulfilled. Provided as a Kubernetes Operator, Theodolite allows defining, sharing, and archiving benchmarks and experiment configurations in declarative files. We demonstrate Theodolite's benchmarking method and show how researchers and cloud engineers can execute existing scalability benchmarks or define new ones with Theodolite.**

***Index Terms*—benchmarking, scalability, cloud-native**

I. Introduction

Cloud-native applications constitute a recent trend for building and operating large-scale software systems [1]. Key concepts are containers, immutable infrastructure, and microservices to build resilient, manageable, and observable software systems [2]. Supported by major cloud vendors, an entire ecosystem of tools has emerged for simplifying, accelerating, and securing the development and operation of software systems in the cloud. Most prominent among these tools is probably Kubernetes, which has become the de-facto standard orchestration tool for cloud-native applications [3]. Scalability is often mentioned as a key driver for adopting cloud-native software architectures [4].

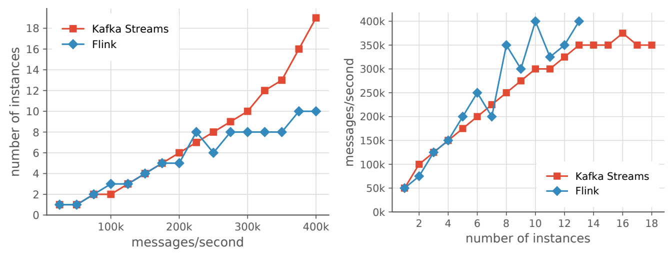
II. The Theodolite Benchmarking Framework

Theodolite[1] is a framework for benchmarking the scalability of cloud-native applications, running in Kubernetes. It automates the benchmarking process by deploying the system under test (SUT) to a Kubernetes cluster, generating load on the SUT, and collecting performance metrics during load generation. Theodolite comes in the shape of a Kubernetes Operator being installed inside the Kubernetes cluster. This significantly improves usability and reproducibility as it allows benchmarks and their executions to be defined in declarative files, which can be written and managed using established Kubernetes tooling [5].

III. Benchmarking Method

Theodolite adopts established definitions of scalability in cloud computing for its benchmarking method [6]. It quantifies scalability by running isolated experiments for different load intensities and provisioned resource amounts, which assess

[1]https://www.theodolite.rocks



(a) Resource demand

(b) Load capacity

Fig. 1: Scalability of two stream processing engines benchmarked with Theodolite's scalability metrics [7].

whether specified SLOs are fulfilled. Two metrics are available: The *demand* metric describes how the amount of minimal required resources evolves with increasing load intensities, while the *capacity* metric describes how the maximal processable load evolves with increasing resources. Hence, both metrics are functions as plotted in Fig. 1.

The terms load, resources, and SLOs are consciously kept abstract as Theodolite leaves it to the benchmark designer to define what type of load, resources, and SLOs should be evaluated. For example, horizontal scalability can be benchmarked by varying the amount of Kubernetes Pods, while vertical scalability can be benchmarked by varying CPU and memory constraints of Pods.

To balance statistical grounding and time-efficient benchmark execution, Theodolite comes with different heuristics for evaluating the search space of load and resource combinations. Other configuration options include the number of repetitions, the experiment and warm-up duration, as well as the amount of different load and resource values to be evaluated.

IV. Executing Benchmarks

Theodolite distinguishes between *benchmarks* and *executions* of benchmarks. Benchmarks describe the deployment of the SUT, a load generator, and potential middlewares used for the benchmark. Additionally, it defines supported load dimensions (e.g., requests per second), resource dimensions (e.g., number of Pod replicas), and SLOs (e.g., 99th percentile latency should be below some threshold). Benchmarks are designed and provided, for example, by standardization organizations, researchers, or within a company (see Section VI).

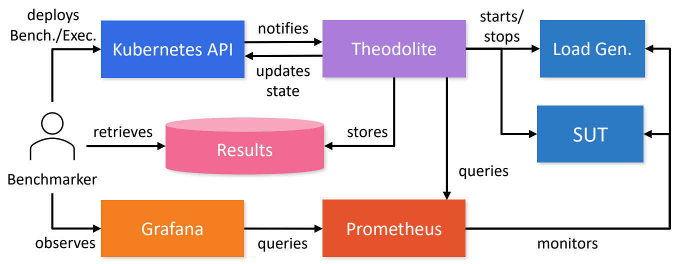


Fig. 2: Interactions between the benchmarker and Theodolite's benchmarking components.

To execute a benchmark, benchmarkers need to create an *execution* resource in Kubernetes. This is usually done by writing a YAML file of kind *execution*, which refers to a benchmark and defines the experimental setup. It selects one of the benchmark's load and resource dimensions as well as other experiment settings such as the scalability metric, a search heuristic, the experiment duration, or the number of repetitions. The execution file can be versioned and archived to support verifiability and repeatability of scalability evaluations.

Fig. 2 depicts the general benchmarking process. Benchmarkers use the *kubectl* command line tool to deploy both the benchmark and the execution to the Kubernetes API server. According to the Kubernetes Operator Pattern, Theodolite is notified of new executions. Depending on whether another benchmark is executed at the moment, Theodolite either starts executing the new benchmark or queues it for later execution. Based on the provided execution and the associated benchmark, Theodolite runs experiments for different combinations of load intensity and resource amounts. In each such experiment, Theodolite starts the SUT and the load generator with a configuration modified according to the load and resources to be tested. During the entire benchmark execution, Prometheus collects monitoring data provided by the SUT and the load generator. Once the configured experiment duration has passed, Theodolite stops the SUT and the load generator, queries collected monitoring data from Prometheus, and assesses whether the benchmark's SLOs are fulfilled. While executing a benchmark, Theodolite updates the state of executions in the Kubernetes API to provide feedback to the benchmarker. Theodolite includes Grafana for observability.

Depending on the selected metric, Theodolite eventually creates a CSV file providing the resource demand for each evaluated load intensity or the load capacity for each evaluated resource amount. Additionally, Theodolite stores all raw measurements used for the SLO assessment in CSV files. These can be used for an in-depth analysis with provided Jupyter notebooks. Fig. 1 shows example outputs of these notebooks.

V. Designing Benchmarks

To create a new benchmark, benchmark designers must create a *benchmark* YAML file. This file defines the SUT, the load generator, and infrastructure components (e.g., middlewares) as sets of Kubernetes resources. Theodolite supports arbitrary Kubernetes resources such as *Deployments*, *Services*, or *PersistentVolumes*, which are packaged in *ConfigMaps*. This makes it easy to benchmark systems, for which Kubernetes definitions are already available. Note that this does also not make any restriction on the load generator to be used. Theodolite does not integrate a load generator, but instead can use any load generator, which is provided as a set of Kubernetes resources.

Load and resource dimensions are each defined by *patchers*. Essentially, patchers are functions that take a numerical value as input and modify a Kubernetes resource file, based on the supplied value. For example, to specify the number of Pod replicas as resource dimension, a patcher may be used that sets the *replica* field of a configured Kubernetes *Deployment* to the supplied value.

Benchmarks can provide multiple SLOs. Each SLO is defined declaratively, consisting of a *PromQL* query to retrieve collected monitoring data, functions to aggregate these data over time and over multiple repetitions, and a threshold to check the aggregated monitoring data against.

VI. Benchmarks for Distributed Stream Processing Engines

Theodolite comes with a set of benchmarks for distributed stream processing engines [8]. These benchmarks serve typical use cases for analyzing Industrial Internet of Thing sensor data such as writing measurements to a database or performing different types of aggregations on streaming data. Along with a configurable load generator, Theodolite provides benchmark implementations for the stream processing engines Apache Flink, Kafka Streams, Hazelcast Jet as well as other engines, which are supported by the Apache Beam SDK.

VII. Conclusions and Getting Started

Theodolite is a cloud-native scalability benchmarking framework, provided as free and open-source research software (https://github.com/cau-se/theodolite). It can easily be installed in a Kubernetes cluster via Helm. To get started, see our quickstart page: https://www.theodolite.rocks/quickstart.html.

References

[1] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-native applications," *IEEE Cloud Computing*, vol. 4, no. 5, 2017.

[2] Cloud Native Computing Foundation, "CNCF cloud native definition v1.0," 2018. [Online]. Available: https://github.com/cncf/toc/blob/main/DEFINITION.md

[3] ——, "CNCF annual survey 2021," 2022. [Online]. Available: https://www.cncf.io/reports/cncf-annual-survey-2021

[4] N. Kratzke and P.-C. Quint, "Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study," *Journal of Systems and Software*, vol. 126, pp. 1–16, 2017.

[5] S. Henning, B. Wetzel, and W. Hasselbring, "Reproducible benchmarking of cloud-native applications with the Kubernetes Operator Pattern," in *Symposium on Software Performance*, 2021.

[6] S. Henning and W. Hasselbring, "A configurable method for benchmarking scalability of cloud-native applications," *Empirical Software Engineering*, vol. 27, no. 6, 2022.

[7] ——, "How to measure scalability of distributed stream processing engines?" in *International Conference on Performance Engineering*, 2021.

[8] ——, "Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures," *Big Data Research*, vol. 25, 2021.