Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Descartes
research

Update: Descartes Research Group

Samuel Kounev

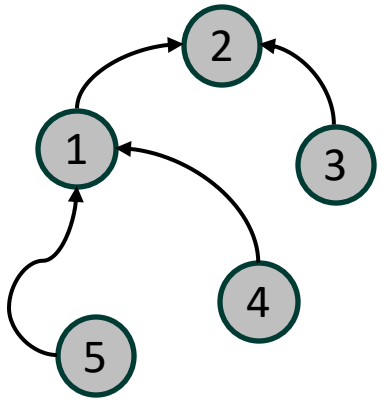
SSP 2025, Kiel, November 4, 2025

Creo: Generating Microservice Applications

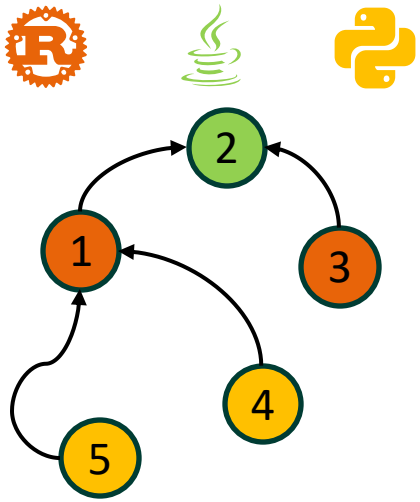
① Inputs

Graph Properties Service Types Languages

② Generation Process



Graph Generator



Graph Colorizer

Source Code Generator

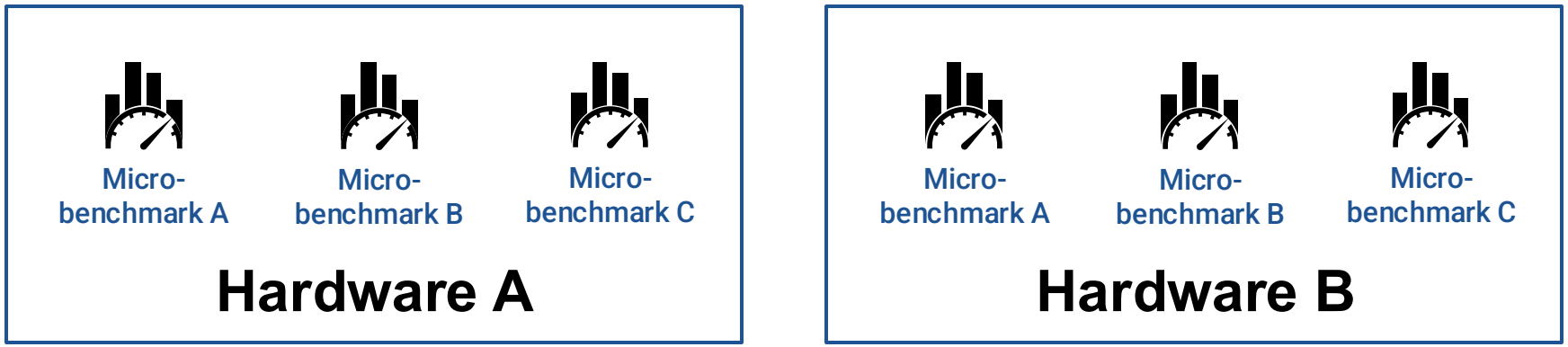
③ Outputs

Source Code Monitoring Load Generation

[1]: Y. Lubas et al. “Generating Executable Microservice Applications for Performance Benchmarking”. In: *ICPE*. 2025, pp. 31–44.

New Project with Snowflake, Inc.

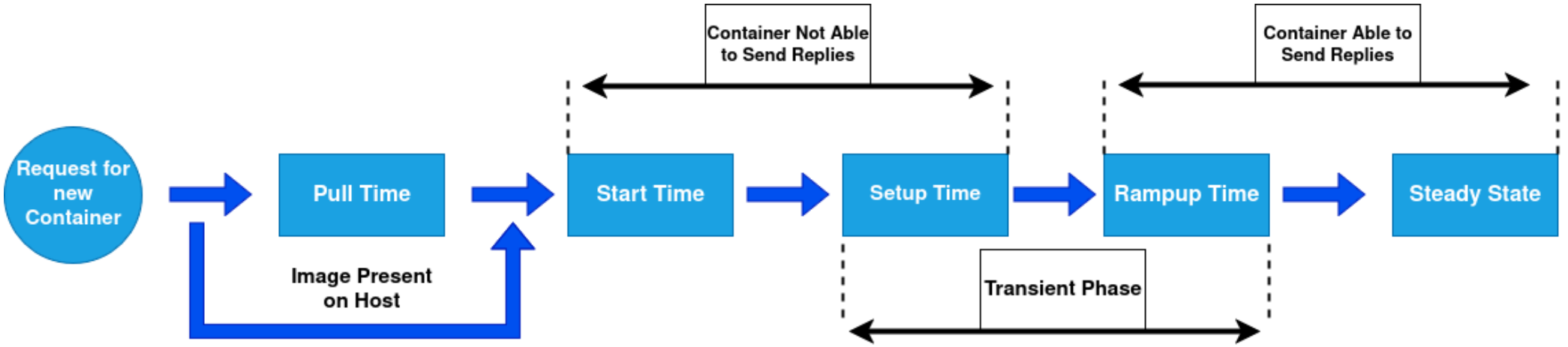
Goal: Predict application-level performance across hardware architectures using microbenchmarks



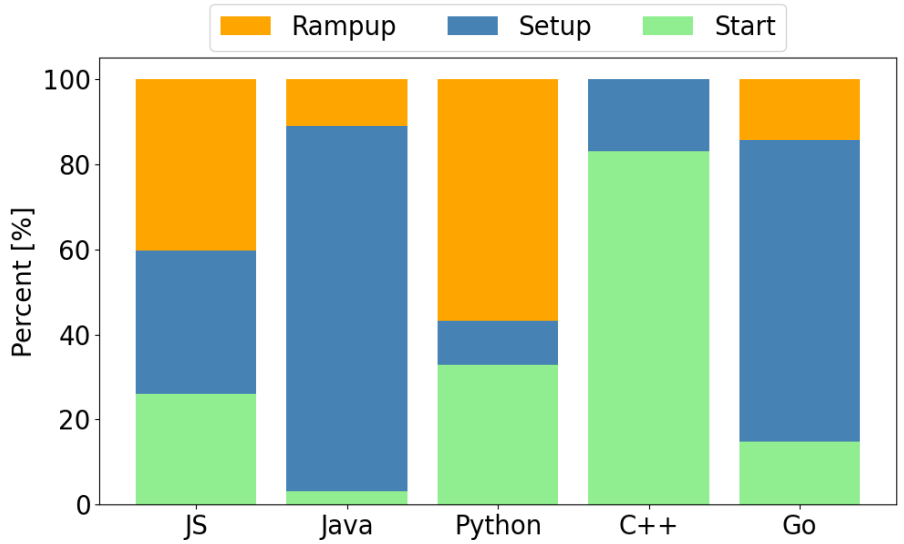
Collect microarchitecture-level metrics (e.g., IPC, cache misses, etc.)

Predict application-level performance based on metrics collected during microbenchmark runs

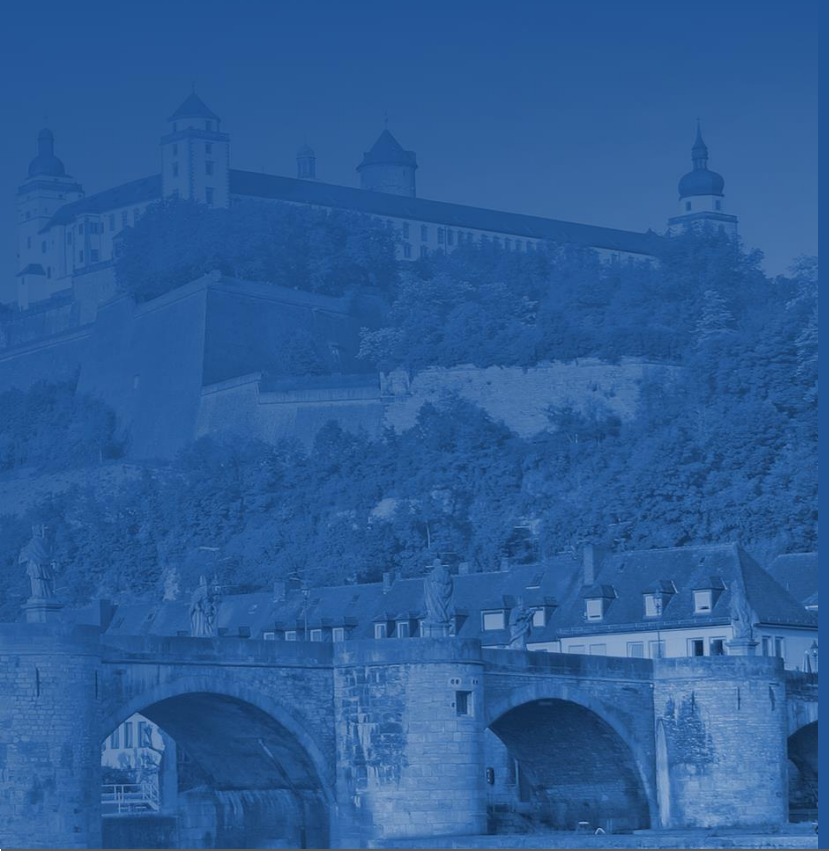
Transient Phases of Microservice Applications



- Scaling and updating cause **frequent starts/stops** of microservice instances
- Problem: Newly started instances often show **unstable performance** -> transient phase
- Our **case study** investigates characteristics of transient phases



[2]: Rohwer, Ivo, et al. "An Empirical Study on Transient Phases of Microservice Applications." To appear in: 2025 MASCOTS.



SOS:
Serverless Scientific Computing and Engineering for Earth Observation and Sustainability Research

Spokespersons: Prof. Dr. Samuel Kounev &
Prof. Dr. Claudia Künzer (Interdis. Projects)

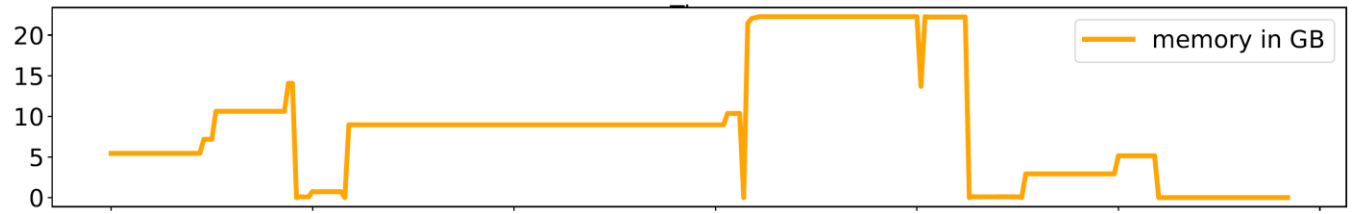
DFG Deutsche Forschungsgemeinschaft
German Research Foundation

DFG Research Unit FOR 5696/1

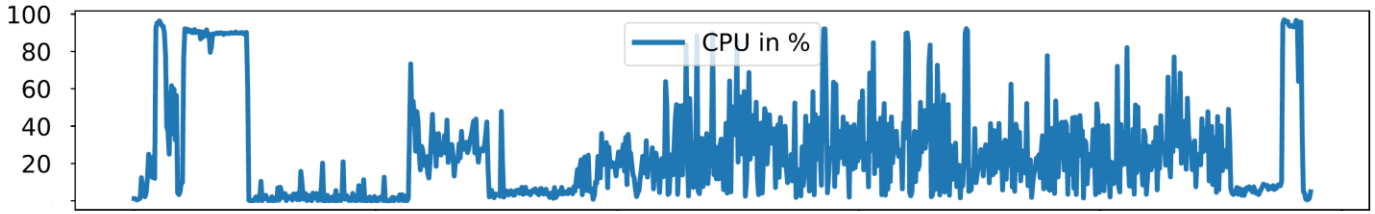
State of Practice

```
# First, we connect to the back-end and authenticate ourselves via Basic authentication.
con = openeo.connect("https://earthengine.openeo.org")
con.authenticate_basic("group11", "test123")

# Now that we are connected, we can initialize our datacube object with the area around Vienna
# and the time range of interest using Sentinel 1 data.
datacube = con.load_collection("COPERNICUS/S1_GRD",
                               spatial_extent={"west": 16.06, "south": 48.06, "east": 16.65, "no
                               temporal_extent=["2017-03-01", "2017-06-01"],
                               bands=["VV"])

# Since we are creating a monthly RGB composite, we need three (R, G and B) separated time range
# Therefore, we split the datacube into three datacubes by filtering temporal for March, April a
march = datacube.filter_temporal("2017-03-01", "2017-04-01")
april = datacube.filter_temporal("2017-04-01", "2017-05-01")
may = datacube.filter_temporal("2017-05-01", "2017-06-01")

# Now that we split it into the correct time range, we have to aggregate the timeseries values i
# Therefore, we make use of the Python Client function `mean_time`, which reduces the time dimen
# by taking for every timeseries the mean value.

mean_march = march.mean_time()
mean_april = april.mean_time()
mean_may = may.mean_time()

# Now the three images will be combined into the temporal composite.
# Before merging them into one datacube, we need to rename the bands of the images, because othe
# they would be overwritten in the merging process.
# Therefore, we rename the bands of the datacubes using the `rename_labels` process to "R", "G"
# After that we merge them into the "RGB" datacube, which has now three bands ("R", "G" and "B")

R_band = mean_march.rename_labels(dimension="bands", target=["R"])
G_band = mean_april.rename_labels(dimension="bands", target=["G"])
B_band = mean_may.rename_labels(dimension="bands", target=["B"])

RG =
RGB

# L
# t
# W

dat

# W

job
job
```

MultiSar



Sentinel 5P



Programming practice:

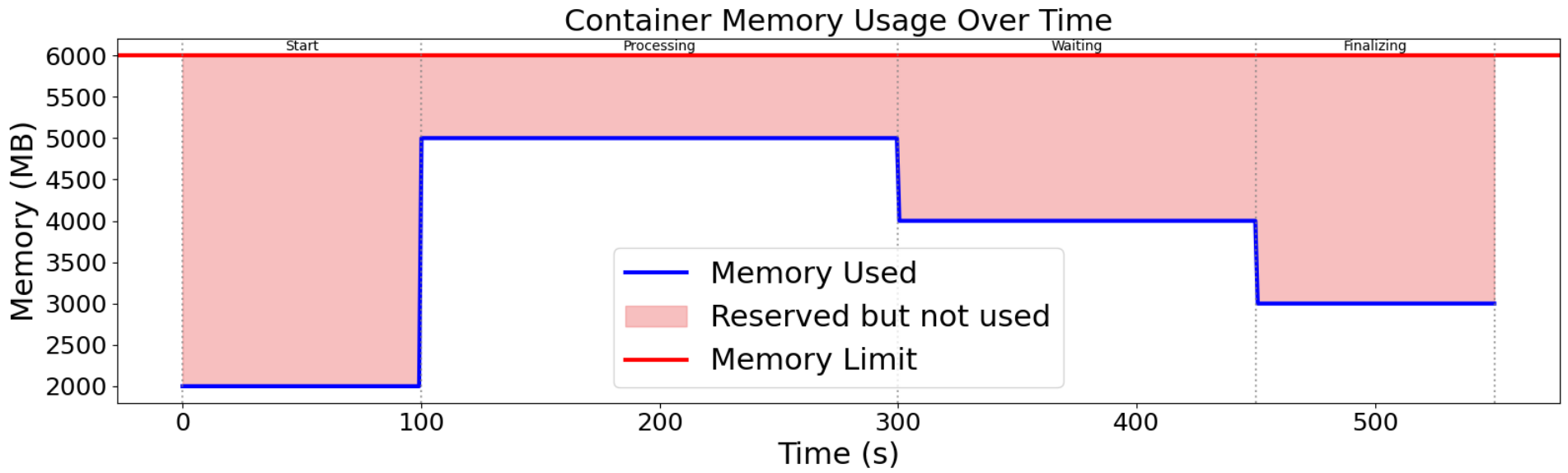
- monolithic jobs
- HW dependent scripts
- low degree of automation
- limited reusability and maintainability

Execution characteristics:

- blackbox workflows (not modeled as DAGs)
- distinct phases within jobs
- varying parallelism & resource demands
- runtimes in the order of hours

Preliminary Work

- In HPC/HDPA systems **resource utilization** is often relatively low
- Reason (among others): Often, **too many resources are reserved** for individual jobs
- Solution: **Phase-aware** resource reservations



[3]: Rohwer, Ivo, et al. "PARAGRAPH: Phase-Aware Resource Demand Profiling for HPDA/HPC Jobs." In: *ICPE*. 2025 , pp. 31–44.

SSP 2025 Talks

A Case Study on the Value of Simulated and Synthetic Microservice Applications for Performance Model Training

A Case Study on the Value of Simulated and Synthetic Microservice Applications for Performance Model Training

Yannik Lubas[1], Martin Straesser[1], Ivo Rohwer[1], André Bauer[2], Samuel Kounev[1]
[1]University of Würzburg, Germany
[2]Illinois Institute of Technology, Chicago, IL, USA

Abstract

Microservices have become a central architectural style for cloud-based systems, driving increased interest in their performance characteristics. However, the limited availability of real-world microservice applications for academic research constrains empirical studies in this area, particularly those using application-agnostic, machine learning-based approaches. Prior work has explored synthetic generation and simulation as alternative means of obtaining performance data; however, their benefits when used alongside real-world measurements are less understood. Using the saturation predictor Monitorless in a case study, we evaluate two scenarios. Our results indicate that generated and simulated applications improve predictive performance on previously unseen, real-world systems. In our case, generated applications reduced false positives, simulated applications enhanced robustness, while their combination yielded the most balanced predictor. The findings suggest that synthetic and simulated applications can effectively complement real-world data, enabling more diverse and robust evaluation of performance models.

1 Introduction

Microservices are an integral part of modern cloud systems, offering scalability and flexibility during development and operation. Yet, their distributed nature also introduces substantial challenges for performance management. Due to the heterogeneous landscape of cloud applications, combined with the fast-paced development of applications enabled by modern DevOps practices, most approaches rely on machine learning (ML) models. Instead of learning specialized models for a single service or application, these models extract application-agnostic performance models that encompass the performance characteristics of multiple, diverse services or applications, limited only by the diversity of the training data.

However, obtaining sufficiently diverse training data remains difficult. Open-source microservice applications suitable for performance studies are scarce, and setting up test environments is a labor-intensive process. Consequently, many research works rely on a limited set of applications, a recurring threat to validity that has been highlighted in the literature [4]. Microservice simulators and synthetic application generators offer a promising way to overcome this limitation by enabling rapid creation of diverse training data.

Building on our prior work, which showed the benefits of augmenting training data with generated applications for performance degradation prediction, this paper presents a case study on enriching training data with both simulated and synthetic applications. We demonstrate that such enrichment improves predictive performance on a dataset of previously *unseen* and *real* applications.

2 Foundations and Previous Work

MiSim [3] is a simulation tool designed to model and analyze microservice applications, with a focus on resilience and performance. Currently, the tool simulates the processing of requests using their CPU consumption. Other resources, such as memory or disk usage, are not simulated yet.

Creo [4] is a multi-language microservice application generator focusing on performance benchmarking. The generated applications are fully executable and have configurable performance characteristics. The built-in support for monitoring, load generation, and deployment automates most of the setup steps for performance experiments.

Monitorless [2] is an ML-based model that predicts instance saturation using platform-level metrics (i.e., CPU usage). The application-agnostic approach relies on the diversity and representativeness of the training data.

In our previous work [4], we demonstrated that Monitorless can benefit from training data enriched with measurements from generated applications. The extended training dataset improved the results on the test dataset (comprising a *real* microservice application).

3 Experiment Design

A major challenge in evaluating application-agnostic approaches, such as Monitorless, lies in constructing representative datasets. Since open-source applications are both rare and complex to deploy and benchmark, experiments with real applications are typically

Towards Intelligent Performance Data Analytics With Graph Databases

Towards Intelligent Performance Data Analytics With Graph Databases

Ivo Rohwer[1], Martin Straesser[1], Yannik Lubas[1], Samuel Kounev[1], André Bauer[2]
[1] University of Würzburg, Germany [2] Illinois Institute of Technology, USA

Abstract

Continuous observability of microservices requires monitoring traces and resource consumption, generating vast amounts of data that are difficult to interpret due to complex interdependencies. While tools like Prometheus, Jaeger, and Grafana support label- and time-based queries, they lack graph-based semantic querying, which is well-suited to model microservice architectures. This paper explores the potential of leveraging the graph database Neo4j for performance data, utilizing its expressive graph query language, Cypher. Using pattern matching, Cypher supports flexible querying of data in various semantic relationships. Our approach will enable users to analyze time series from various observability data sources, such as Prometheus and Jaeger, supporting flexible pattern queries, aggregations, statistical analysis, and temporal as well as structural queries within the microservices context.

1 Introduction

Microservices are a common architectural style for cloud applications, dividing large, complex systems into smaller, more manageable services. Besides their benefits, such as scaling services independently, they introduce new design challenges, such as avoiding bottlenecks and anti-patterns. For this purpose, often graph-based algorithms are used that require a graph data model [1, 2, 3, 4]. The application of graph algorithms relies on semantic graph databases such as Neo4j. While these databases enable advanced graph queries, they are not well-suited for the storage and querying of time series data [7].

In contrast to static graph analysis, microservice performance problems are often identified during operation by monitoring resource consumption, traces, and error rates. Standard tools for collecting and visualizing traces and metrics include OpenTelemetry, Prometheus, Jaeger, and Grafana.

Regarding graph analysis and observability tools, we identify a gap between these two areas: Observability tools typically provide graph-based visualizations, but lack a comprehensive graph data model that enables graph queries such as Cypher in Neo4j. In this paper, we discuss our vision for an extension of Neo4j that enables the querying of time series data from observability tools via Neo4j. It offers a variety of aggregation and analysis functions that are typically found in time series query languages, but are lacking in graph databases. We aim to facilitate the combination of traditional graph and time series queries, reflecting the complex nature of microservice observation data in terms of temporal sequences and graph relations.

2 Related Work

Ammar et al. [7] presented HyGraph, which aims to combine semantic graphs and time series databases in a native manner. They demonstrated the advantages of this concept using a prototype consisting of Neo4j and TimescaleDB. In contrast, our approach is tailored to the observability domain and aims to incorporate both traces and time series. Conversely, zero-ETL graph systems such as PuppyGraph [8] allow observability data to be queried using Cypher without loading it into a graph database. While these solutions allow temporal relations to be queried to a certain extent, they lack specific functions for querying, aggregating, and analysing time series.

Several publications [1, 2, 3, 4] address the use of graph models to analyze the architectural quality of microservice systems, but they lack consideration of metrics and traces. Several machine learning approaches [5, 6] to root cause detection rely on graph modeling of traces and metrics. However, these approaches do not involve query systems such as Neo4j, but only the input representation of neural networks.

Current, well-established observability tools, such as Grafana, Kiali, and Apache SkyWalking, offer graph visualisations, but not graph queries as enabled by Cypher. SkyWalking uses a query language called GraphQL, but this only refers to the structuring of the queries and does not utilize an underlying graph structure in the queried data, despite including some topology functions. On the other hand, TraceQL from Grafana Tempo allows pattern matching, similar to Cypher, but only for traces.

3 Approach

We aim to combine graph queries with performance metric and trace queries. The Neo4j engine should execute these queries, but metrics and traces should

Questions?

Descartes research

www.descartes-research.net