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

remain stored in classic observability tools such as Prometheus or Jaeger. To this end, we plan to implement a Neo4j plugin that enables the querying of performance data from external data sources via Cypher. In addition, our plugin should include various other functions, such as the application of time series analysis and comparison, as well as functions for searching graph elements based on temporal constraints. Neo4j supports the close integration of Java plugins into the database engine. Specifically, it enables user-defined procedures invoked after the Cypher keyword `CALL` and user-defined functions. In the following, we provide an overview of our assumptions and planned functions, which are currently at various stages of implementation.

Data Storage/Sources. We aim to query time series and traces from different data sources (including the option of locally stored time series, but not recommended). After registering a data source, such as Jaeger or Prometheus, with the plugin, from the user's perspective, it should make no difference whether the requested data is stored locally or in external time series databases.

Graph Model. In related work, various graph models are used to represent microservice systems [4]. We aim to represent microservice architectures using service nodes and operation nodes to represent the endpoints of the services. This allows for detailed analysis of call chains. The system state is mapped as follows: A graph node is created for each deployment unit (e.g., a Kubernetes pod) and connected to the corresponding service. These nodes can then be associated, for example, with resource consumption, latency, or error rate time series.

Time Series Queries. The Cypher query language is designed for searching in graphs using pattern matching. We plan to extend this with Neo4j procedures, enabling the querying of time series related to individual nodes, in a similar way to conventional time series databases, such as Prometheus or InfluxDB.

Aggregation Functions. Similar to the query languages of typical time series databases, aggregation functions such as the moving average, integral, or sum will be applicable to time series. Neo4j already offers many aggregation functions at the node level.

Time Series Analysis Functions. Alongside the aggregation functions, we plan to incorporate various analysis functions, such as change point detection.

Mathematical Relations. As well as semantic relations, it would also be beneficial to query mathematical relations, such as correlations between time series.

Temporal Search. To find pods from a specific time period, for instance, we also intend to develop procedures to facilitate searching for objects. Appropriate indexing is necessary to enable efficient time-based searching in the graph database.

Structural Search. While Cypher enables very flexible graph queries, we plan to implement procedures to shortcut frequently used structural graph search queries. For example, the plugin should make it easier to query all operations that are directly or indirectly called by an initial operation, or all operations that directly or indirectly call a service.

4 Examples

To demonstrate our Neo4j extension for querying microservice traces and metrics, we will utilize the GAIA dataset[1], which CloudWise has published. This dataset contains simulated tracing and metric data from the MicroSS sample application, which consists of five microservices: webservice, redisservice, mobservice, dbservice, and logservice.

Below, we provide a few examples that reference the MicroSS architecture to illustrate how our Neo4j extension can be applied. We present a classical Cypher query, followed by a time series query, and then a combined query. These examples are based on preliminary implementations and may not reflect the final syntax of our Neo4j extension.

Call Chain Discovery. In this first example, we apply a classical Cypher query. Suppose we want to know the impact of an error in the `dbservice.db_login_methods` operation of the `dbservice` service on the other services in the MicroSS example system. The following query can be used to identify all services that directly or indirectly call this operation:

```
MATCH(targetOp:Operation
    {name: "dbservice.db_login_methods"})
MATCH (caller:Service)-
    [:HAS_OPERATION]->(:Operation)-
    [:DEPENDS_ON*0..]->(targetOp)
RETURN DISTINCT caller
```

Time Series Query and Aggregation. The following query demonstrates a time series query with our *timegraph.data.get_time_series* procedure:

```
MATCH (p:Pod {name : "webservice1"})
CALL timegraph.data.get_time_series(p,
   "cpu_total",
   {time : "2021-07-01T10:54:43Z",
   range : "+2h",
   aggregation : "integral"})
YIELD timestamps, values
RETURN timestamps, values
```

Using a syntax similar to that of time series database query languages, this query retrieves the CPU consumption over two hours for the pod named `webservice1` and aggregates the results using the integral function.

Error Localization. The next example illustrates the combination of graph and time series queries: Let us assume that we observe an unusual error rate during the `logservice.login_model_implement` operation of the `logservice` service and want to locate

[1] https://github.com/CloudWise-OpenSource/GAIA-DataSet

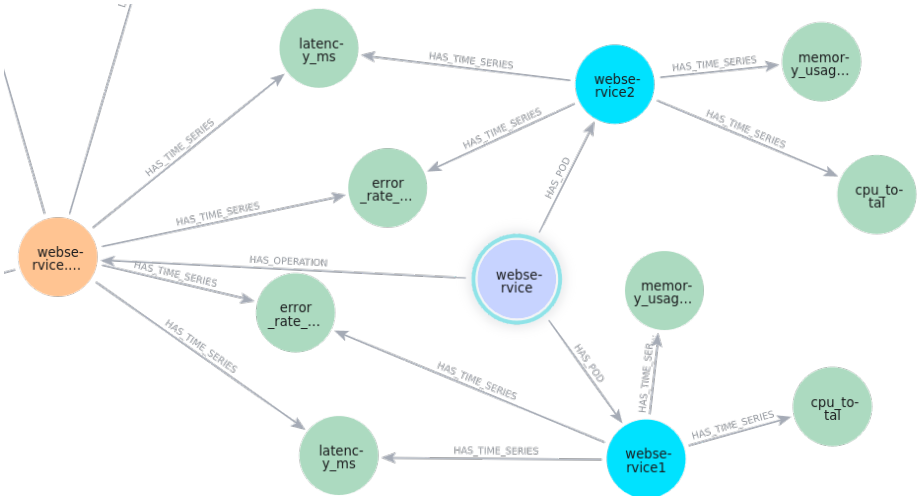

Figure 1: Illustration of the data model associated with an example service. Node types: service representation (purple), operation representation (orange), pod representation (blue), and time series data (green).

the cause. We can then use the following query to identify all operations in the microservice representation graph that are called directly or indirectly by the `logservice.login_model_implement` operation, and return the corresponding error rates.

```
CALL timegraph.search.
    getCalledOperationsFromOperation(
    "logservice",
    "logservice.login_model_implement")
YIELD operation
CALL timegraph.data.get_time_series(
    operation, "error_rate_pct")
YIELD timestamps, values
RETURN operation, timestamps, values
```

Here, we identify the cause of errors in a call chain by combining graph-based search and time series queries.

5 Conclusion

Microservice architectures and deployments are often modeled using graph representations. Current frameworks used for querying microservice traces and metrics, such as Grafana, Kiali, and SkyWalk, usually provide graph visualizations but lack graph queries in the style of semantic graph databases. This paper presents our vision for graph-based querying of microservice traces and metrics. It utilizes the Neo4j semantic graph database and extends its functionality to enable the querying of time series and traces from various observability data sources, including Prometheus and Jaeger. Our planned framework will include functions for temporal and structural searches of objects within the context of microservice applications, as well as analysis functions. By doing so, we intend to achieve a level of query flexibility that surpasses the capabilities of existing tools and approaches.

Acknowledgements

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) – 510552229.

References

[1] R. S. Ashley Sun. *How Lending Club Manages Microservices with Neo4j*. Neo4j Blog. accessed: 2025-08-19. Oct. 2015.

[2] S.-P. Ma et al. “Using service dependency graph to analyze and test microservices”. In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. 2018.

[3] E. Gaidels and M. Kirikova. “Service Dependency Graph Analysis in Microservice Architecture”. In: *Perspectives in Business Informatics Research*. Ed. by R. A. Buchmann et al. Cham: Springer International Publishing, 2020.

[4] A. Tiwari. “Unveiling Graph Structures in Microservices: Service Dependency Graph, Call Graph, and Causal Graph”. In: *Abhishek Tiwari* (2024).

[5] S. Zhang et al. “No more data silos: Unified microservice failure diagnosis with temporal knowledge graph”. In: *IEEE Transactions on Services Computing* (2024).

[6] Z. Zhao et al. “CHASE: A Causal Hypergraph based Framework for Root Cause Analysis in Multimodal Microservice Systems”. In: *arXiv preprint arXiv:2406.19711* (2024).

[7] M. Ammar et al. “Towards hybrid graphs: Unifying property graphs and time series”. In: *28th International Conference on Extending Database Technology*. 2025.

[8] S. Wang. *Logs, Metrics, Traces: From Theory to Use at Coinbase*. PuppyGraph blog. accessed: 2025-08-25. July 2025.