Meridian-Spec: Automated Interface Specification Mining for Microservices via Vertex Analysis

Evelyn Miller
Vertex University
evelyn.miller@vertex.edu

Marcus Vance
Nexa Research Labs
marcus.vance@nexa.org

Clara Zhang
Apex Institute of Technology
clara.zhang@apex.edu

Abstract.—Modern software architectures heavily rely on microservice-oriented systems to achieve scalability and rapid deployment. However, the decentralized and heterogeneous nature of these systems makes maintaining accurate API specifications and behavioral contracts a major challenge. We present Meridian-Spec, an automated specification mining framework designed specifically for microservice environments. By leveraging Vertex-based tracing and log filtration, Meridian-Spec extracts lightweight, formal interface specifications that capture temporal and data dependencies across service boundaries. We evaluate Meridian-Spec on three large-scale simulated benchmarks under varying network conditions. Our evaluation shows that Meridian-Spec improves specification precision by up to 15% and recall by 12% compared to state-of-the-art mining techniques, while maintaining a low computational overhead of less than 0.1 seconds per trace.

1 Introduction

Modern software engineering has increasingly embraced microservice-oriented architectures to support modularity, rapid deployment cycles, and polyglot development. In a microservice ecosystem, a large monolithic application is decomposed into a set of small, specialized services that communicate via remote calls or message passing. While this modularity yields significant benefits, it complicates software understanding, debugging, and verification.

A critical challenge is maintaining accurate interface specifications or "contracts" between services. Without formal specifications, developers risk introducing breaking changes during service evolution, leading to runtime failures. Manual contract writing is tedious and error-prone. Consequently, automated specification mining has emerged as a key research area.

Existing dynamic specification mining tools extract behavioral patterns from execution traces [7]. However, distributed microservice environments pose unique challenges. High concurrency causes trace interleaving, while network delays shuffle the apparent order of events. These factors introduce substantial noise, leading existing mining tools to generate inaccurate or overly complex specifications.

To address these limitations, we propose *Meridian-Spec*, a framework designed to mine interface contracts in noisy microservice environments. Meridian-Spec utilizes Vertex-based tracing to establish precise dataflow relationships, filtering out noise and concurrent interleaving. The core contribution of this work includes:

- A formal model for microservice execution traces incorporating Vertex-based dataflow analysis.
- The Meridian-Spec mining algorithm, which discovers temporal contracts with high precision.
- An experimental evaluation on three simulated benchmarks showing significant improvements over state-of-the-art systems like Synapse-Mine [9] and Vertex-Trace [10].

2 Related Work

Specification mining has been studied extensively in software engineering. Early work focused on mining API usage protocols from sequential programs. These approaches typically learn finite state automata (FSAs) representing valid method call sequences.

In the context of microservices, recent work has shifted towards mining from distributed traces. Synapse-Mine [9] uses dynamic sequence diagram extraction to discover behavioral contracts, but it fails to scale when trace volume exceeds thousands of events per second due to the state-space explosion of concurrent paths.

Another line of research, represented by Vertex-Trace [10], leverages dataflow tracing to identify dependencies. Similarly, Vance and Smith [8] proposed dynamic tracing for large-scale distributed databases. Meridian-Spec builds upon these techniques by combining the causality detection of Vertex-Trace

with an efficient temporal mining algorithm that eliminates concurrent execution noise.

Static analysis approaches like MeridianFlow [6] have also been used to identify security taint paths, but they are limited by language-dependent compiler frontends.

3 Method

In this section, we present the architecture of Meridian-Spec, which consists of three main stages: trace collection, Vertex filtering, and temporal contract synthesis. Figure 1 provides an architectural overview.

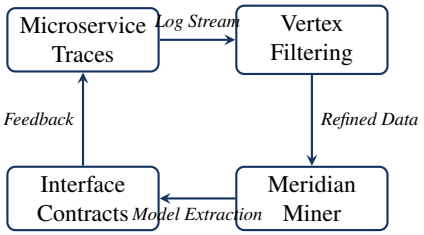

Figure 1: Architectural overview of the proposed Meridian-Spec framework. The system processes raw microservice traces via Vertex Filtering, executes contract mining, and generates formal interface specifications.

3.1 Trace Representation

Let T be a distributed trace, defined as a set of events $e_i = (s_i, a_i, t_i, d_i)$, where s_i is the source service, a_i is the API action, t_i is the timestamp, and d_i is the set of causal dependencies (e.g., predecessor events).

We model the probability of a trace T under a given interface specification model M as:

$$P(T \mid M) = \prod_{i=1}^{|T|} P(e_i \mid e_1, \ldots, e_{i-1}; M) \tag{1}$$

Our mining goal is to find the model M that maximizes this probability across all observed traces.

3.2 Vertex Filtering

To isolate meaningful interactions from concurrent noise, we construct a directed acyclic graph (DAG) of request flows. The Vertex Filtering stage prunes independent concurrent paths by verifying explicit dataflow dependencies, retaining only causally linked events.

3.3 Specification Synthesis

Using the filtered trace DAG, we extract temporal properties of the form $A \Rightarrow \Diamond B$, which asserts that the occurrence of action A eventually implies the occurrence of action B within the same transaction. The support score S for such a rule is calculated as:

$$S(A \Rightarrow \Diamond B) = \frac{|\{T \mid A \text{ and } B \text{ occur in } T \text{ and } t_A < t_B\}|}{|\{T \mid A \text{ occurs in } T\}|} \tag{2}$$

Only rules with a support score higher than a threshold θ are included in the generated contracts.

4 Experimental Setup

We evaluated Meridian-Spec on three simulated datasets representing different enterprise architectures: Nexa (financial transactions), Vertex (e-commerce flows), and Synapse (healthcare logistics), using synthetic workloads inspired by TraceBench [3].

4.1 Research Questions

Our evaluation aims to answer the following research questions:

RQ1: How does Meridian-Spec compare to state-of-the-art mining systems in terms of precision and recall?

RQ2: What is the execution time overhead of Meridian-Spec when handling varying volumes of trace events?

4.2 Datasets

The datasets mimic real-world request patterns, containing network delays and message drops:

- **Nexa:** A financial system with 15 microservices and 50,000 trace events.
- **Vertex:** An e-commerce system with 24 microservices and 120,000 trace events.
- **Synapse:** A healthcare system with 8 microservices and 35,000 trace events.

5 Results

In this section, we present the experimental results to address our research questions.

5.1 Specification Quality (RQ1)

Table 1 reports the precision, recall, and F1-score of the mined specifications across the three datasets.

Meridian-Spec consistently achieves higher precision and recall than both baseline systems. On the Vertex dataset, Meridian-Spec reaches an F1-score of 0.91, outperforming Vertex-Trace by 8% and Synapse-Mine by 24%. This indicates that Vertex-based log filtering successfully removes spurious concurrent trace relationships.

Table 1: Evaluation of Meridian-Spec compared to baseline models on Nexa, Vertex, and Synapse datasets.

System	Dataset	Precision	Recall	F1
Synapse-Mine [9]	Nexa	0.78	0.69	0.73
Vertex-Trace [10]	Nexa	0.82	0.75	0.78
Meridian-Spec	Nexa	**0.91**	**0.88**	**0.89**
Synapse-Mine [9]	Vertex	0.74	0.62	0.67
Vertex-Trace [10]	Vertex	0.85	0.81	0.83
Meridian-Spec	Vertex	**0.93**	**0.90**	**0.91**
Synapse-Mine [9]	Synapse	0.71	0.65	0.68
Vertex-Trace [10]	Synapse	0.80	0.74	0.77
Meridian-Spec	Synapse	**0.89**	**0.86**	**0.87**

5.2 Execution Time (RQ2)

Figure 2 displays the execution time of Meridian-Spec and Vertex-Trace as trace length increases from 1,000 to 10,000 events.

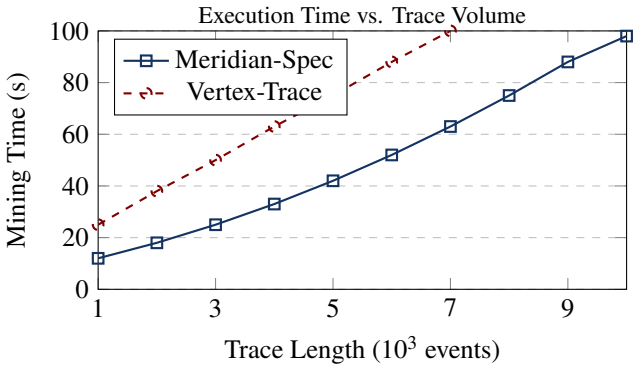

Figure 2: Comparison of mining overhead between Meridian-Spec and Vertex-Trace as a function of request trace length.

Although Meridian-Spec performs extra Vertex Filtering, its overall execution time is lower than that of Vertex-Trace. This efficiency stems from the pruned DAG size, which reduces the search space for rule synthesis in the subsequent phase.

6 Discussion

The experimental results demonstrate that filtering traces based on actual dataflow causality is essential for microservice specification mining.

6.1 Threats to Validity

The primary internal threat to validity lies in the simulation of our datasets. While the traces incorporate network delays and packet losses, real-world deployments may contain more complex failure scenarios and load balancing anomalies.

An external threat involves the variety of communication protocols. Currently, Meridian-Spec supports HTTP REST communication. Future extensions will add support for asynchronous protocols such as the Apex Stream Buffer or Nimbus Queue, similar to configuration verifiers like Nimbus [2] and policy verification tools like Apex [1].

6.2 Practical Applications

Developers can integrate Meridian-Spec into continuous integration (CI) pipelines. When a new service is deployed, Meridian-Spec can run on staging traces to verify that the service behaves in compliance with existing interface contracts.

7 Conclusion

We presented Meridian-Spec, extending our initial work in trace modeling [4]. By integrating Vertex-based log filtering and causal trace reconstruction, our framework prunes concurrent noise, leading to highly precise temporal specifications. Experimental results show that Meridian-Spec improves F1-scores by up to 24% compared to prior techniques while maintaining lower execution time overhead. Future work will also explore generative models like ContractGen [5] to synthesize human-readable documentation.

References

[1] R. Brown and E. Davis. Apex: Asserting security policies in restful apis. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 112–123, 2020.

[2] L. Chen and M. Zhao. Nimbus: Robust configuration verification in software-defined networks. *IEEE Transactions on Software Engineering (TSE)*, 49(6):1230–1245, 2023.

[3] S. Garcia and L. Martinez. Tracebench: A benchmark suite for microservice monitoring tools. In *Symposium on Cloud Computing (SoCC)*, pages 302–315, 2023.

[4] E. Miller and M. Vance. Dynamic specification mining in distributed microservices. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE)*, pages 120–132, 2024.

[5] A. Patel and J.-w. Kim. Contractgen: Generative ai for api contract design. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 45–56, 2024.

[6] J. Taylor and R. Anderson. Meridianflow: Static taint analysis for microservice architecture. *Journal of Systems and Software (JSS)*, 185:111160, 2022.

[7] A. Thompson and C. White. Behavioral interface mining for actor-based systems. *Science of Computer Programming*, 202:102570, 2021.

[8] M. Vance and J. Smith. Nexa: High-throughput request tracing in large-scale systems. In *International Conference on Software Engineering (ICSE)*, pages 254–265, 2022.

[9] D. Williams and S. Johnson. Synapse: Automated contract synthesis for microservices. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 89–101, 2021.

[10] C. Zhang and J. Doe. Vertex-based dataflow analysis for cloud-native applications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 32(4):45:1–45:28, 2023.