The Tale of Errors in Microservices: Extended Abstract

I-Ting Angelina Lee
Washington University in St. Louis
Department of Computer Science & Engineering
St. Louis, Missouri, USA
angelee@wustl.edu

Zhizhou Zhang
Uber Technologies Inc.
Programming Systems Group
Sunnyvale, California, USA
zzzhang@uber.com

Abhishek Parwal
Uber Technologies Inc.
Marketplace Configurations
Sunnyvale, California, USA
abhishek.parwal@uber.com

Milind Chabbi
Uber Technologies Inc.
Programming Systems Group
Sunnyvale, California, USA
milind@uber.com

Abstract

Microservice architectures have become the de facto paradigm for building scalable, service-oriented systems. Although their decentralized design promotes resilience and rapid development, the inherent complexity leads to subtle performance challenges. In particular, ***non-fatal*** errors — internal failures of remote procedure calls that do not cause top-level request failures — can accumulate along the critical path, inflating latency and wasting resources.

In this work, we analyze over 11 billion RPCs across more than 6,000 microservices at Uber. Our study shows that nearly 29% of successful requests experience non-fatal errors that remain hidden in traditional monitoring. We propose a novel ***latency-reduction estimator*** (***LR estimator***) to quantify the potential benefit of eliminating these errors. Our contributions include a systematic study of RPC error patterns, a methodology to estimate latency reductions, and case studies demonstrating up to a 30% reduction in tail latency.

CCS Concepts

• **Software and its engineering → Cloud computing**; • **Computing methodologies → Modeling methodologies**; • **Computer systems organization → Reliability**.

Keywords

Microservices, Non-fatal Errors, Latency Estimation, Critical Path Analysis, RPC Errors, Performance Optimization

ACM Reference Format:
I-Ting Angelina Lee, Zhizhou Zhang, Abhishek Parwal, and Milind Chabbi. 2025. The Tale of Errors in Microservices: Extended Abstract. In *Abstracts of the 2025 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS Abstracts '25), June 9–13, 2025, Stony Brook, NY, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3726854.3727320

1 Introduction

Microservices decompose monolithic applications into small, independently deployable services. Each service handles a distinct business functionality and communicates via Remote Procedure Call (or RPC for short) mechanisms (e.g., REST, gRPC, Thrift, YARPC). Although this architecture supports agile development and scalable deployment, it also results in complex call graphs.

A single user request may trigger hundreds of RPCs. Although many of these RPCs succeed, our data shows that a substantial fraction fail but remain ***non-fatal*** — the top-level request still completes with a success code. Such "hidden" errors often arise from best-effort calls or repeated lookups that fail for a portion of requests. Developers frequently overlook these errors because they do not show up in the *final* error metrics. Yet, these errors often appear on the ***critical path*** [3], the longest sequence of dependent RPCs in the request, inflating tail latencies and wasting resources.

At Uber, more than 6,000 microservices interact in a dynamic, large-scale ecosystem catering to hundreds of millions of users. Our distributed tracing system, Jaeger, samples over 840 million traces daily; logging each RPC within the sampled request to provide a detailed view of internal RPC behavior. Despite an overall RPC error rate of only 0.9%, our analysis reveals:

- Approximately 29% of successful requests contain at least one non-fatal error.
- Nearly 84% of user-exposed endpoints experience non-fatal errors, while only 16% remain consistently error-free.

Such observations motivate the need for a systematic approach to quantify and mitigate the impact of non-fatal errors.

2 Key Contributions

Our work makes the following key contributions:

(1) **Comprehensive Characterization:** We systematically study RPC error patterns in large-scale microservices and demonstrate that non-fatal errors are widespread and correlate strongly with increased tail latency. By analyzing more than 52 million traces and 11 billion RPCs, we provide detailed insights into error propagation, error types, and their impact on performance.
(2) **Latency-Reduction Estimator:** We introduce a novel methodology to simulate an "error-eliminated" execution by zeroing out erroring RPCs while preserving causal dependencies.

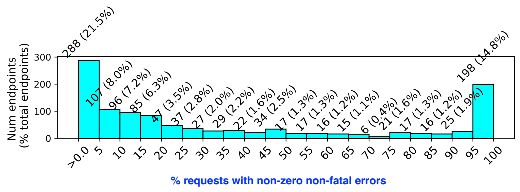

Figure 1: Distribution of non-fatal errors over endpoints.

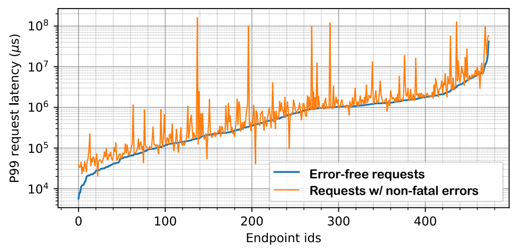

Figure 2: Latency is higher for requests with non-fatal errors compared to that of error-free requests.

This approach yields an upper bound on the latency reduction potential.

(3) **Practical Use Cases:** We present case studies — such as mobile app launch optimization, user customization failures, and parallel data fetch inefficiencies — that demonstrate how eliminating redundant or guaranteed-failure RPCs can reduce tail latency, reduing the up to . In one case, we reduced the latency by 30%.

3 Observations

Our analysis leads to several important observations:

- **Observation 1:** Most endpoints (84%) experience a non-trivial fraction of requests with non-fatal errors.
- **Observation 2:** Requests with non-fatal errors often contain multiple RPC errors, even when the overall failure rate is low.
- **Observation 3:** The majority of non-fatal errors cluster around four key types: entity not found, aborted, failed precondition, and resource exhausted.
- **Observation 4:** Non-fatal errors typically originate deeper in the RPC call graph than fatal errors and exhibit short propagation lengths.
- **Observation 5:** Internal APIs exhibit a bimodal resiliency distribution—most either completely stop or propagate errors.
- **Observation 6:** Requests with non-fatal errors perform more work, on average having 1.9× more RPCs than error-free requests.
- **Observation 7:** Latency in requests with non-fatal errors is significantly higher (1.8× median, 2.9× P99) compared to error-free requests.
- **Observation 8:** The LR estimator indicates a significant potential for latency reduction across many endpoints and APIs.
- **Observation 9:** Non-fatal errors disproportionately impact tail latency more than overall latency.

Figure 1 displays a histogram correlating the fraction of requests with non-fatal errors (x-axis) to the number of endpoints (y-axis). The first histogram bin shows that 288 endpoints had $(0, 5\%]$ of their gateway requests containing at least one non-fatal error. As the error rate increases, the number of affected endpoints decreases and then spikes again, where 198 endpoints have over 95% of their requests containing non-fatal errors—indicating these endpoints consistently operate with errors.

Figure 2 compares the tail (99th percentile) latency of error-free requests with those that have non-fatal errors. The x-axis represents different endpoints, and the y-axis is the 99^{th} percentile of request latency. The blue line shows error-free requests, whereas the orange line shows requests with non-fatal errors. The P99 latency of requests with non-fatal errors is, on average, 2.9× more than the P99 latency of error-free requests.

4 The LR estimator

A key challenge in mitigating the impact of non-fatal errors is determining which endpoints warrant deeper investigation. Straightforward approaches do not yield the desired outcome. For instance, simply targeting endpoints with the highest number of non-fatal errors is ineffective, since not all non-fatal errors affect overall latency. Similarly, one can focus on endpoints with the most non-fatal errors on the critical path; however, a larger number of non-fatal errors does not necessarily translate into higher latency. Finally, one might measure the impact of non-fatal errors on the critical path using a method similar to [3]. Yet this, too, can be problematic because eliminating erroneous RPCs might alter the critical path, such that the resulting latency does not shrink proportionally to the duration of the eliminated erroring RPCs from the original critical path.

We propose a **latency-reduction estimator** (LR estimator) to quantify the latency impact of non-fatal errors. The LR estimator computes the hypothetical latency of an execution if all non-fatal RPCs were eliminated while preserving the causal dependencies among RPC calls. The LR estimator operates under two key assumptions:

(1) **Zero Duration for Errors:** All RPCs that return errors in the observed execution are assumed to have zero duration in the error-eliminated execution.
(2) **Preservation of Dependencies:** The causal relationships among RPC calls are maintained. This ensures that inherent sequence and delays dictated by program semantics remain even when error durations are removed.

By applying these assumptions to a Jaeger trace, the LR estimator simulates a new "error-eliminated" timeline and recomputes the critical path of a request. The difference between the observed latency and the hypothetical latency represents the maximum potential improvement achievable by mitigating non-fatal errors.

Unlike naive methods that might subtract the duration of erroring RPCs, the LR estimator is cognizant of erroring RPCs occurring off the critical path. Moreover, eliminating an error may cause shifts in the critical path—sometimes revealing new bottlenecks. The LR estimator recursively adjusts the timeline, ensuring that:

- The relative ordering of dependent RPCs is preserved.
- The minimal work required by parent services (before initiating a child RPC) remains intact.
- Any shift in the critical path is recalculated, providing a realistic estimate of latency reduction.

% latency reduction	Number of endpoints				
	Tail-1	Tail-5	Tail-10	Tail-20	Tail-50
(10, 20]	62	62	60	57	60
(20, 30]	24	22	22	26	25
(30, 40]	12	11	11	8	6
(40, 50]	8	10	9	8	13
(50, 60]	7	6	7	9	3
(60, 70]	7	7	7	6	8
(70, 80]	5	5	5	5	7
(80, 90]	6	6	6	5	2
(90, 100]	1	1	1	2	3

Table 1: Endpoint count with tail latency-saving potentials.

Using LR estimator, we performed analyses on a large set of traces from production. Table 1 bins the endpoints by their saving potential at different tail latencies. Tail-1% shows how much the worst 1% (averaged from 99th to 100th percentile) of the latency can be reduced by eliminating errors in those traces. Tail-5, Tail-10, Tail-20, and Tail-50 are defined similarly. We notice that 62 endpoints can reduce their Tail-1% latency by 10-20%; 24 endpoints can reduce their Tail-1% latency by 20-30%; cumulatively, 26 endpoints can reduce their Tail-1% latency by $> 50\%$. Similarly, 60 endpoints can reduce their Tail-50% latency by 10-20% and 25 endpoints can reduce their Tail-50% latency by 20-30%. $> 50\%$.

The LR estimator provides engineers with a robust and practical tool to prioritize optimization efforts by precisely quantifying the latency improvements achievable by mitigating non-fatal errors across diverse endpoints and internal APIs. In contrast to previous work, such as LatenSeer [2], which also performs "what-if" analyses on distributed traces, the LR estimator offers two critical advantages. First, unlike LatenSeer's probabilistic approach, which provides limited guarantees, the LR estimator employs a deterministic methodology that rigorously maintains causal relationships. Second, the LR estimator avoids the expensive, repeated training required by LatenSeer, which can take hours per endpoint; instead, LR estimator incurs zero training cost and completes analysis rapidly (e.g., only $1.5\mu s$ for a large trace with 10,000 RPCs). Additional details, proofs, and evaluations can be found in our full paper [1].

5 Case Studies and Outcomes

Our methodology was applied to multiple endpoints and internal APIs at Uber. Notable use cases include:

- **App-Launch Optimization:** The mobile app launch workflow is critical for our company due to its high usage and stringent latency requirements. A request to the `app-launch` gateway endpoint triggers numerous RPCs for state-machine transitions, database queries, and other services. The LR estimator analysis identified the `app-launch` endpoint as a major latency contributor, revealing a potential 27.8% reduction in P99 latency. Flame graphs in Figure 3 showed that two RPCs—`pool-provider` and `internal-provider`—consistently failed in tail requests. Although the overall request succeeded, these failures inflated latency. Further investigation revealed that while most requests were correctly handled by `internal-provider`, about 3% were misrouted to an `external-provider` due to lingering code from a discontinued pooling feature. This unnecessary invocation of the `pool-provider` RPC, which was guaranteed to fail, was responsible for significant delays.
  By eliminating the redundant `pool-provider` call, the team achieved approximately a 30% reduction in tail latency for the `app-launch` endpoint, with related endpoints also benefiting from up to a 10% latency improvement.
- **User Customization Failure:** For the *get-stores-view* endpoint, redundant calls to *get-user* led to errors that accounted for 57% of the P99 latency. A proposed solution is to implement request-level caching to avoid repeated error invocations.
- **Parallel Data Fetch Optimization:** An internal API (*fetchInfo*) issued concurrent queries to two data stores. In practice, the calls were executed sequentially, leading to 50% wasteful work. Optimizing the routing of these requests based on context (mobile vs. web) can yield significant improvements.

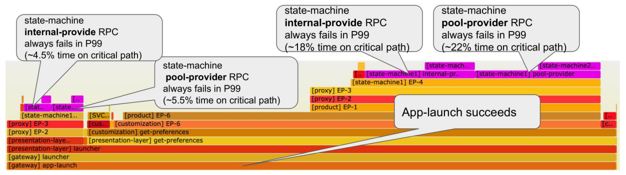

Figure 3: `pool-provider` and `internal-provider` RPCs fail in the P99 profiles on two different call paths but `app-launch` succeeds. Wasteful `pool-provider` call consumes 27.8% of the critical path.

6 Conclusions

Our study of non-fatal errors in Uber's microservice ecosystem reveals that, although these errors do not cause outright request failures, they incur a substantial latency penalty, especially at the tail end of the distribution. The proposed latency-reduction estimator quantifies the potential benefits of eliminating such errors by simulating an error-eliminated execution. Our key contributions include a comprehensive characterization of error patterns, a novel estimation methodology, and real-world case studies demonstrating tangible performance improvements.

In summary, our findings emphasize that proactive error management and targeted optimization — by focusing on the most impactful non-fatal errors — can lead to more efficient, scalable, and responsive microservice architectures. We believe these methods broadly apply beyond Uber's ecosystem, offering valuable insights for any organization facing similar challenges. While this work focuses on non-fatal errors, the LR estimator methodology is general. It can estimate latency changes due to increasing or decreasing the duration of any set of RPCs in a complex microservice graph.

References

[1] I-Ting Angelina Lee, Zhizhou Zhang, Abhishek Parwal, and Milind Chabbi. 2024. The Tale of Errors in Microservices. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 3, Article 46 (Dec. 2024), 36 pages.

[2] Yazhuo Zhang, Rebecca Isaacs, Yao Yue, Juncheng Yang, Lei Zhang, and Ymir Vigfusson. 2023. LatenSeer: Causal Modeling of End-to-End Latency Distributions by Harnessing Distributed Tracing. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '23)*. Association for Computing Machinery, New York, NY, USA, 502–519.

[3] Zhizhou Zhang, Murali Krishna Ramanathan, Prithvi Raj, Abhishek Parwal, Timothy Sherwood, and Milind Chabbi. 2022. CRISP: Critical path analysis of Large-Scale Microservice Architectures. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 655–672.