One-Size-Fits-None: Understanding and Enhancing Slow-Fault Tolerance in Modern Distributed Systems

Ruiming Lu, Yunchi Lu, Yuxuan Jiang, Guangtao Xue, Peng Huang

Preprint

Software

NSDI '25

Motivation

Fail-slow faults (e.g., degraded disks, networks) are subtle and can be impactful to overall system performance.

In the 12 years since the last major study (**limplock, SoCC'13**), systems have evolved drastically:

- **Asynchronous designs** and **event-driven architectures**.
- **Cloud-native deployments** and **dynamic workloads**.
- More complex, high-concurrency **hardware**.

Do modern systems handle slow faults well? We conduct a **systematic study** to find out.

Study Methodology

Evaluated 6 modern distributed systems' fault tolerance under diverse faults.

cassandra, etcd, kafka, CockroachDB, HBase, hadoop

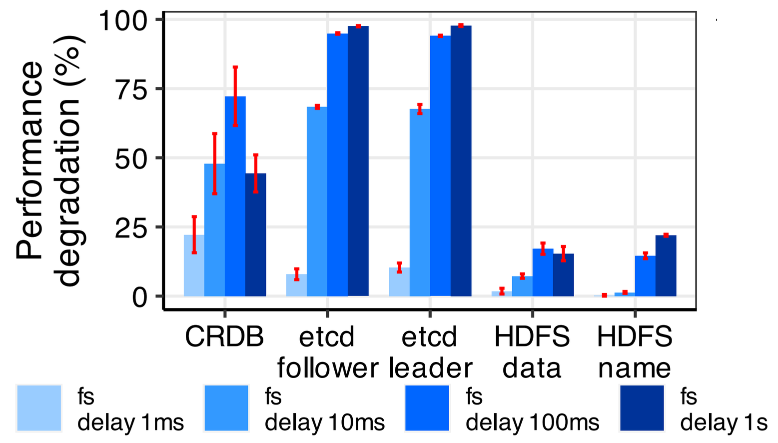
Latest stable versions

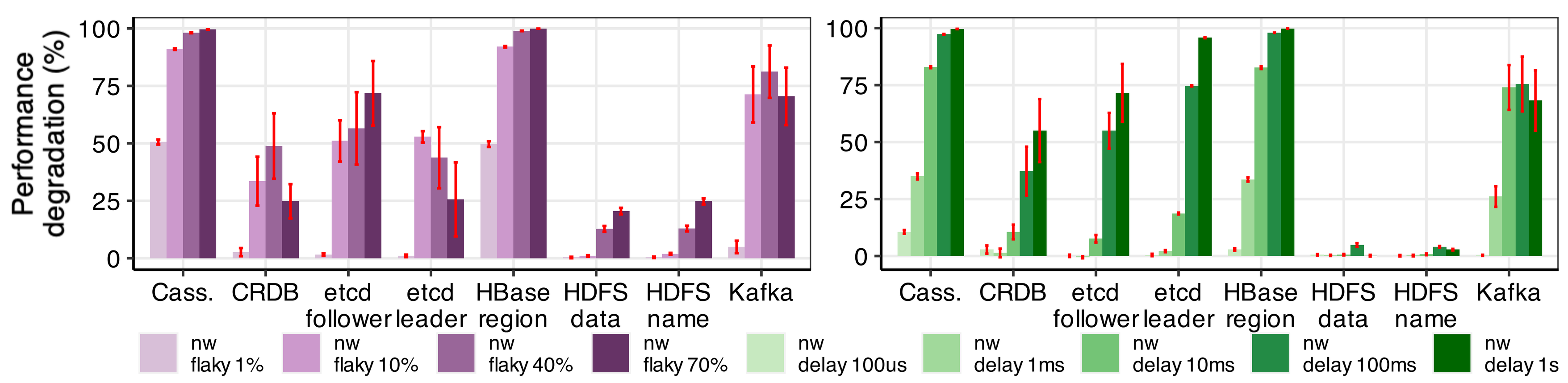
Diverse Kinds of Services

Diverse Benchmarking Workloads

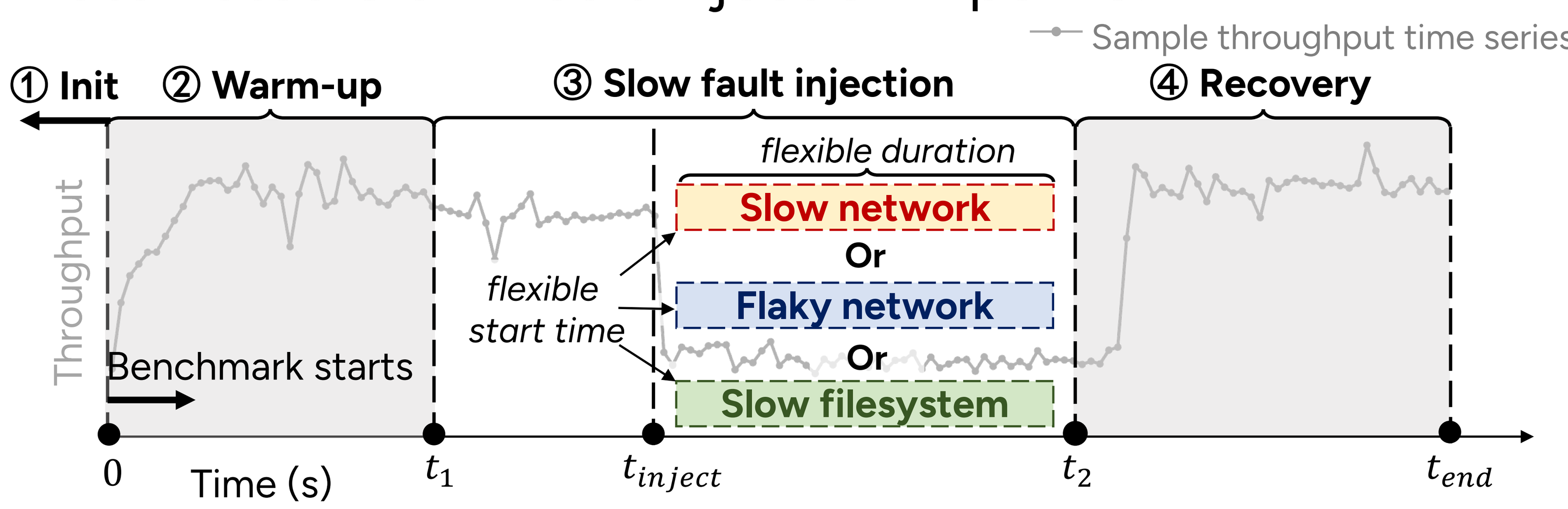
Injected **diverse slow faults**:

- **Network delays** / **packet loss**
- **File system slowness**
- Varying **severity levels**
- Different injected **node roles**





Automated Slow-Fault Injection Pipeline:

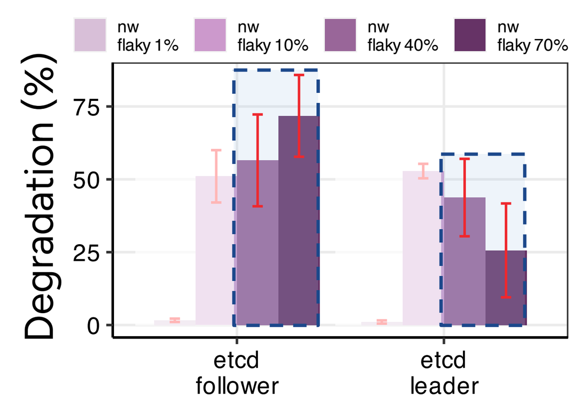


Key Findings (complete list in the paper)

1. Slow-fault tolerance is nuanced and sensitive to
Slow faults: Severity, type, location, duration, start time
Deployment: Resources, configs, workloads

Examples of **Counter-Intuitive Behaviors:**

- ⚡ A **small slow fault** can cause **more degradation** than a larger one.
- ⚡ **Slow followers** can impose **1.5× more performance penalty** than slow leaders.



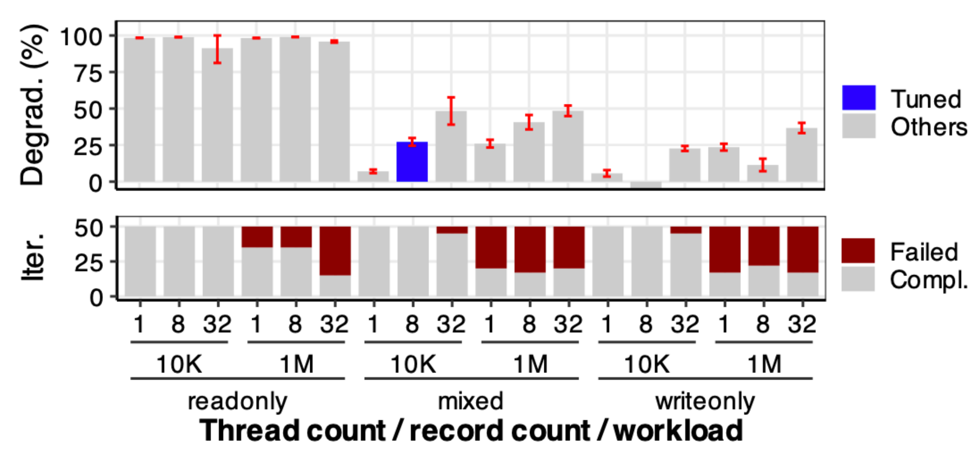
Compared to a slow ***leader****,*
a slow ***follower*** *yields...*
p40 30% higher degradation
p70 177% higher degradation

- ⚡ **Scaling up resources** improves baseline performance, but **amplifies the impact** of slow faults.

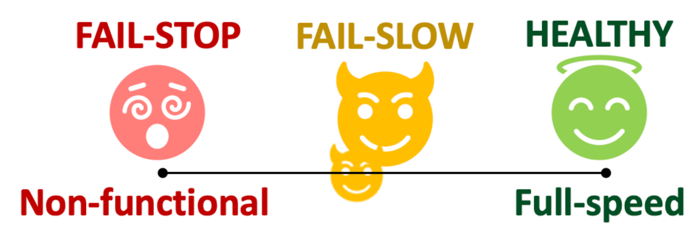
2. (Finding 7) **"Danger zones"** where slight fault increases trigger major performance drops are common across systems.

One-Size-Fits-None: The Flaws of SOTA Approaches

Slow-fault handling mechanisms exist, but detection still **relies on static thresholds**. *(e.g., Copilot [OSDI'20], IASO, HBase, Cassandra configs)*



Finetuned thresholds fail under different workloads.



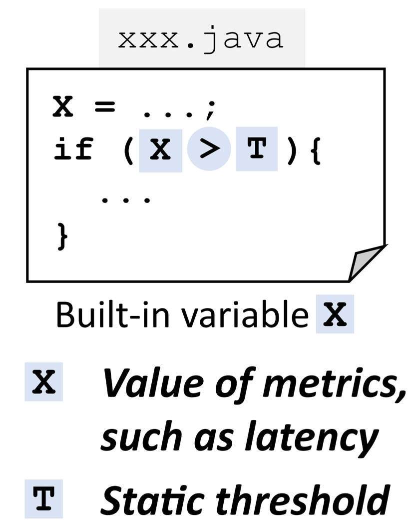
Fail-slow is ***non-binary*** and ***dynamic***

Hard thresholds won't work well!

Relying on *static, fine-tuned* configurations makes a system's slow-fault tolerance *fragile*

ADR – Adaptive Detection at Runtime

Failure detection needs to be ***adaptive***

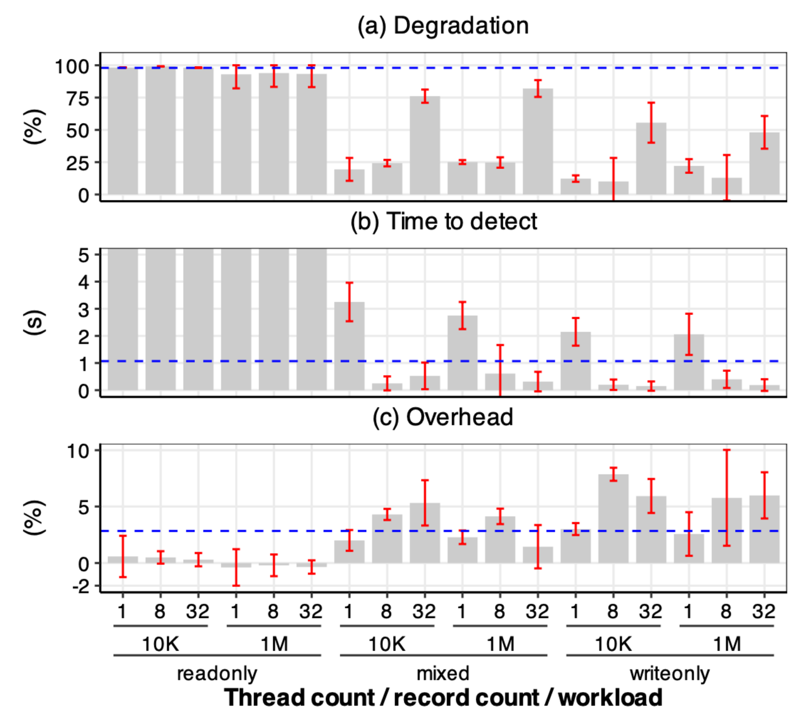


Simply use X's p99 as T's value may not work:

- Always **1% false positives**.
- Fails to distinguish between **workload changes** and real slowness.
- **Infrequent updates** dilute p99, letting true faults slip by.

ADR: *A lightweight, plug-in library for detecting fail-slow issues in distributed systems.*

1. Traces built-in slowness metrics such as latency.
2. Automatically adapts thresholds based on metrics'
   1. *p99*
   2. *Update frequency to rule out normal variations.*



Reduce degradation by 16-90%

Timely detection in seconds

Minimal 2.8% average overhead

Contributions

1. Automated testing pipeline to measure slow-fault tolerance
2. Slow-fault tolerance is nuanced and sensitive to *Slow faults (5 findings) and Deployment (4 findings)*
3. Detecting slowness with static thresholds is insufficient
4. ADR – lightweight, adaptive slow-fault detection library at runtime