Metastable Failures in the Wild

Lexiang Huang$^{1,3,*}$, Matthew Magnusson$^2*$, Abishek Bangalore Muralikrishna$^2$, Salman Estyak$^1$, Rebecca Isaacs$^3$, Abutalib Aghayev$^1$, Timothy Zhu$^1$, and Aleksey Charapko$^2$

*Equal contribution
What are Metastable Failures?

• Example: Retry Storm

Takeaway: Permanent overload even after the trigger is removed
Metastable Failures are Prevalent

- Can be catastrophic
  - E.g., 4 out of 15 major outages in the last decade at AWS

- Ad-hoc diagnosis
  - Persistent congestion
  - Persistent overload
  - Retry storms
  - Death spirals
  - etc.

- Ad-hoc recovery
  - Load-shedding
  - Rebooting
  - Adding more resources
  - Tweaking configurations

Insight: These different-looking failures can be characterized under one taxonomy.
Metastability in the Wild – Survey

• We search through over 600 public post-mortem incident reports
  • Identify 21 metastable failures in
    • Large cloud infrastructure providers
    • Smaller companies and projects

• Can cause major outages
  • 4-10 hours most commonly
  • Incorrect handling leads to future incidents
  • An important class of failures to study
Defining Metastability – System States

Stable

Load increase or capacity decrease

Vulnerable

Overloading trigger

Metastable Failure

Recovery

Sustaining effect

[Bronson et al.]
Survey Summary

• Triggers
  • About 45% are due to engineer errors
    • Buggy configuration or code deployments
    • Latent bugs
  • About 35% are due to load spikes
  • 45% involve multiple triggers

• Sustaining effects
  • Load increase due to retries (over 50%)
  • Expensive error handling
  • Lock contention
  • Performance degradation due to leader election churn
Survey Summary

• Recovery
  • Direct load-shedding
    • Throttling
    • Dropping requests
    • Changing workload parameters
  • Indirect load-shedding
    • Reboots
    • Policy changes
Metastability Taxonomy – Trigger

• One or more events that overload the system
• Two types:

**Load-spike trigger**

**Capacity-decreasing trigger**
Metastability Taxonomy – Sustaining effect

- A feedback loop that keeps the system overloaded
- Two types:

  - **Workload amplification**
    - Amplification continues
    - Trigger applied
    - Trigger fixed
    - Amplification starts

  - **Capacity degradation amplification**
    - Trigger applied
    - Trigger fixed
    - Amplification starts
    - Amplification continues
Four Metastability Scenarios

**Load-spike** trigger

Common incidents due to retries in the survey

**Capacity-decreasing** trigger

Replicated State Machine

Load

Capacity-degradation

Garbage Collection (GC)

Capacity-degradation

Look-aside Cache
Four Metastability Scenarios

**Load-spike trigger**

Common incidents due to retries in the survey

**Capacity-decreasing trigger**

Replicated State Machine

**Garbage Collection (GC)**

**Look-aside Cache**
Metastability due to GC – Sustaining Effect

Load-spike → High queue length

Capacity degradation amplification

High GC behavior

Job processing slows down

- More active objects to process during a GC cycle
- Higher memory pressure causes more GC cycles
- GC causes application to pause and slow down

Sustaining effect: Contention between arriving traffic and GC consuming resources
Metastability due to GC – Timeseries

- **Load-spike** triggers high queue length and high GC behavior
- **Queue** continues building up

![Graph](image)
Metastability due to GC – Timeseries

- **Load-spike** triggers high **queue length** and high **GC behavior**
- **Queue** continues building up
- Aggressive **load-shedding** does not lower the **GC behavior**
Degrees of Vulnerabilities

- **System load** determines vulnerability
  - Tradeoff: Efficiency vs. Vulnerability

Max heap size = **256 MB**
Degrees of Vulnerabilities

- **System load** determines vulnerability
  - Tradeoff: Efficiency vs. Vulnerability

- **System configs** impact vulnerability
  - Larger memory $\Rightarrow$ Lower vulnerability

Max heap size = **256 MB** \hspace{1cm} Increase memory size \hspace{1cm} Max heap size = **384 MB**
Lessons

• Detect and react to trigger quickly to avoid metastable failures
  • Sustaining effects may not be immediate
  • Sustaining effects take time to amplify the overload

• Design systems to eliminate/minimize sustaining effects
  • Common case optimizations may cause or exacerbate sustaining effect
     ➔ Might not be possible to eliminate sustaining effect entirely
     ➔ Consider the slow path, not just the fast path
Lessons

• **Understand the degree of vulnerability of the system to control risk**
  • System load and capacity determines vulnerability
    ➔ Load testing can reveal issues
    ➔ Adding capacity can lower vulnerability

• System config affects vulnerability
  ➔ Control relevant configs to lower vulnerability
Lessons

• Recover from metastable failure by breaking the sustaining effect cycle
  • Fix the triggers to prevent recurrence
    • Negate load spikes by load shedding
    • Rollback or halt deployments
    • Hot-fix software bugs

• End the overload to break the sustaining effect cycle
  • Load-shedding (e.g., admission control, graceful degradation)
  • Increase capacity
  • Change policy to reduce amplification factors
Conclusion

• **Metastable failure** – permanent overload even after triggers are removed

• They are prevalent and can **cause major outages**

• Understanding the **sustaining effects** and the **degree of vulnerability** in systems is critical to prevent metastable failures

• Three open-sourced metastable failure examples
  
  https://github.com/lexiangh/Metastability