Hodor

Detecting and Remediating Overload in LinkedIn Microservices

Conf42 SRE, June 2022

Bryan Barkley, Vivek Deshpande

LinkedIn

“Improve service resiliency by reliably detecting overloads and automatically remediating them while minimizing production impact.”

Our Mission

Agenda

- **Hodor Framework Overview**
- Overload Detectors: CPU, GC and ThreadPool
- Overload Remediation: Adaptive load shedder and Request Retries
- Monitoring and Rollout
- A Success story
- Related/Future work

Hodor Framework

Holistic **O**verload **D**etection and **O**verload **R**emediation

- Hodor is designed to detect service overload caused by multiple root causes, and to automatically remediate the problem by dropping just enough traffic

- Hodor has 3 components:
  - Overload Detectors
  - Load Shedder
  - Platform-specific adapter to combine the two

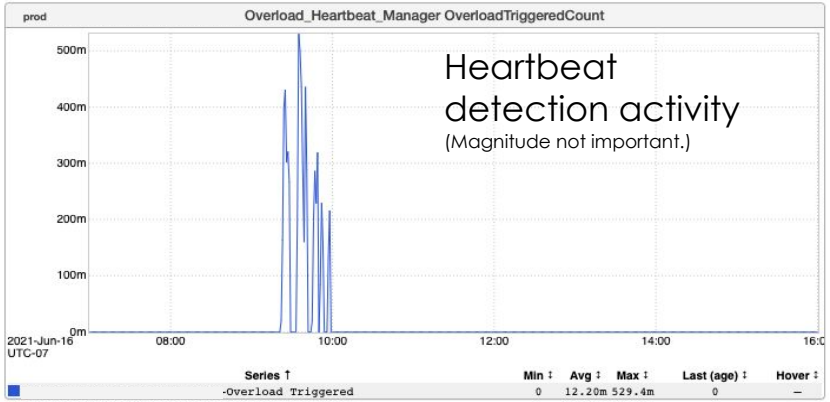
Agenda

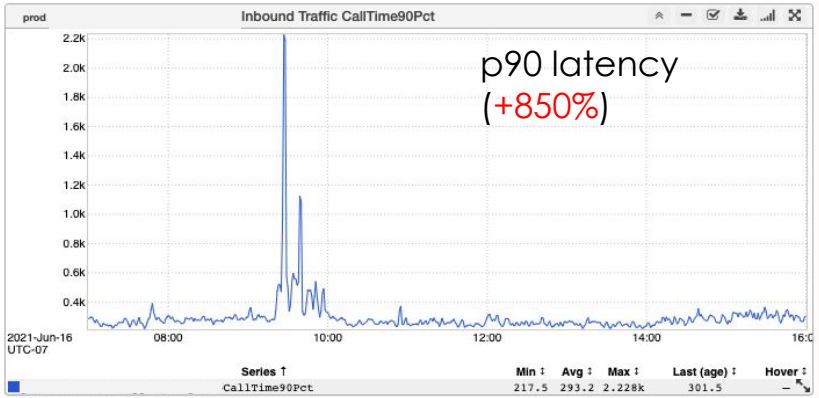
- Hodor Framework Overview
- **Overload Detectors: CPU, GC and ThreadPool**
- Overload Remediation: Adaptive load shedder and Request Retries
- Monitoring and Rollout
- A Success story
- Related/Future work

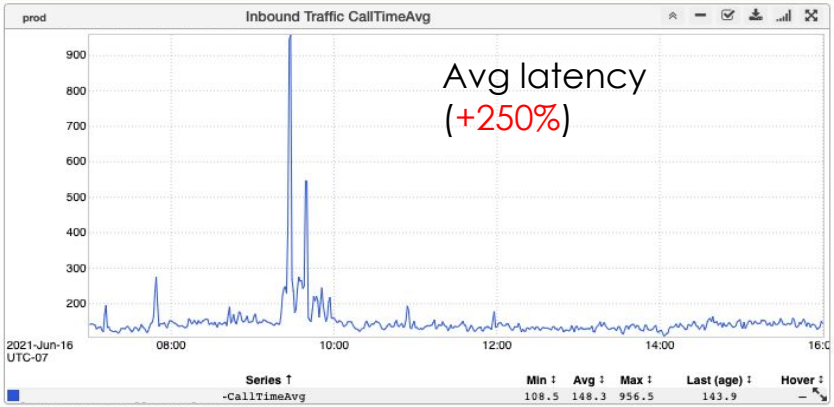
CPU Overload detection - Heartbeat

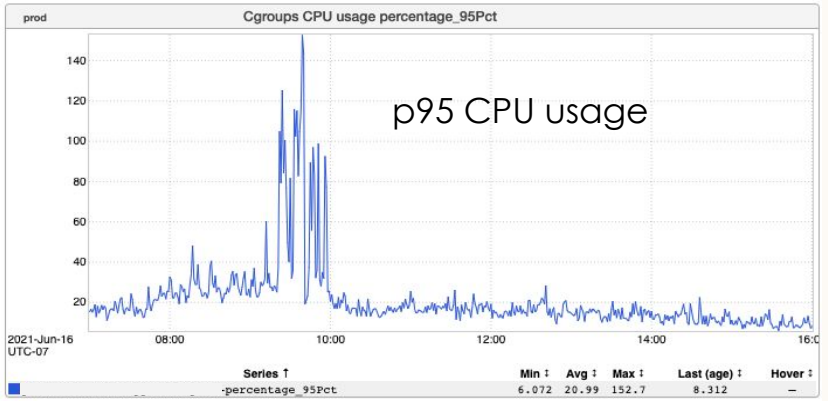
- Objective
  - Early and accurate detection of **CPU bottlenecks**
- Heartbeat algorithm
  - Heartbeat thread: lightweight background thread on a schedule
  - Monitor heartbeat thread wakeups
- Rationale
  - Direct way to measure availability of useful CPU cycles
  - Applicable irrespective of bare-metal, VM, cgroup, cfs quota/shares config etc.

CPU Overload detection case in prod





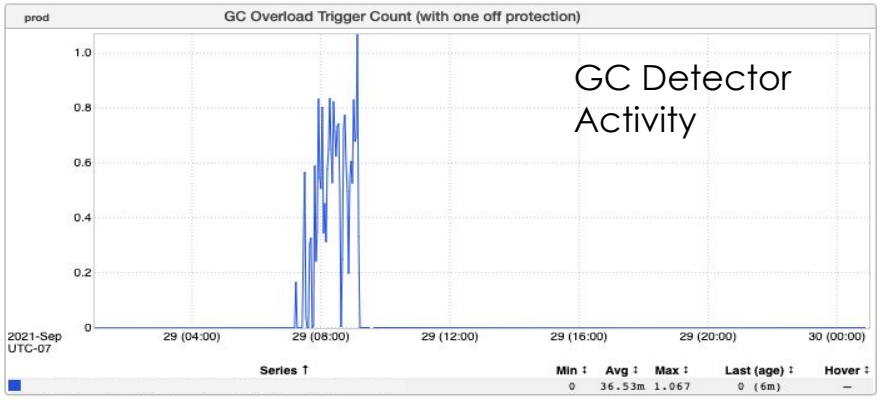




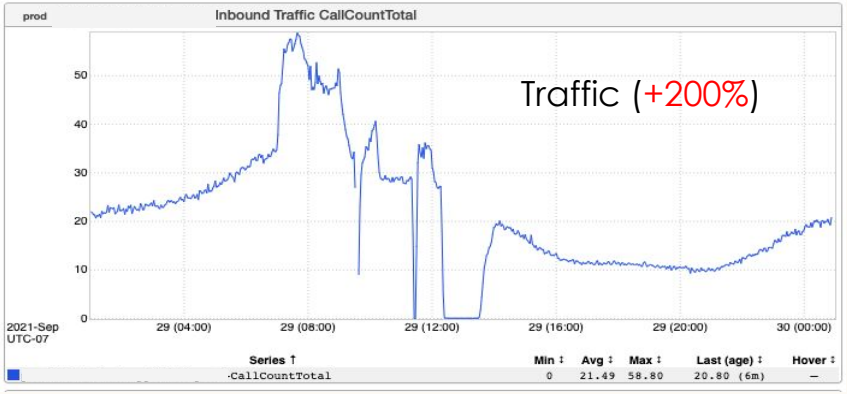
GC Overload detection

- Objective
  - Early and accurate detection of **bottlenecks caused by excessive GC**
- GC Overload Detection algorithm
  - Calculate amortized percentage of time spent in GC over a given time window.
  - The GC Overhead Tiers based on percentage ranges of GC Overhead.
  - If the duration in GC Overhead Tiers exceed the violation period, signal overload
- Rationale
  - A unified signal that catches different types of GC issues.
  - Detect GC overhead before it causes CPU Overload

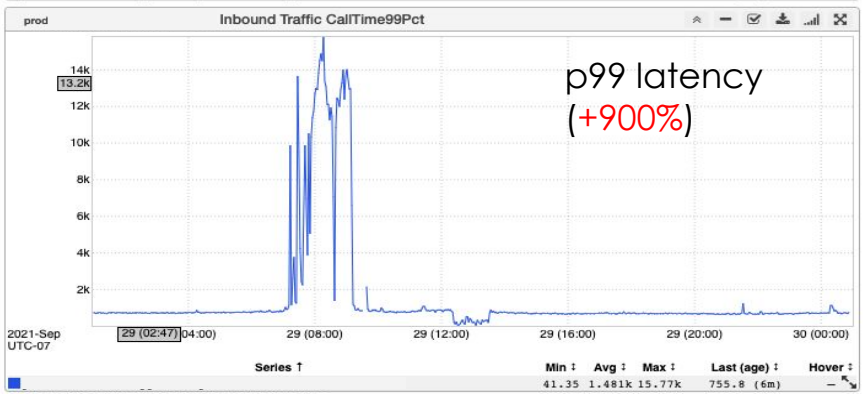
GC Overload detection case in prod



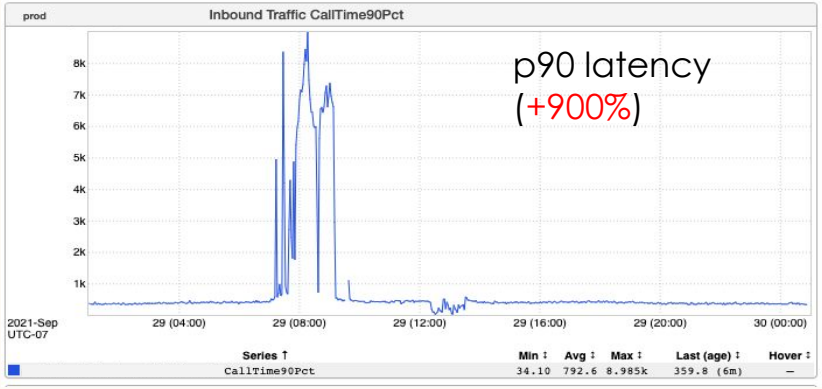
GC Detector Activity



Traffic (+200%)



p99 latency (+900%)

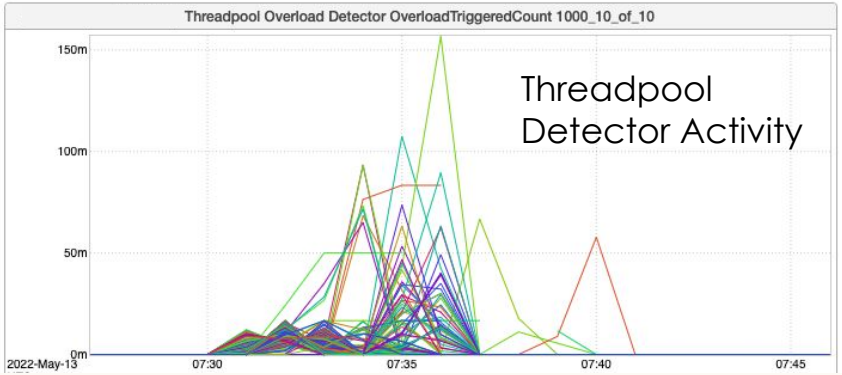


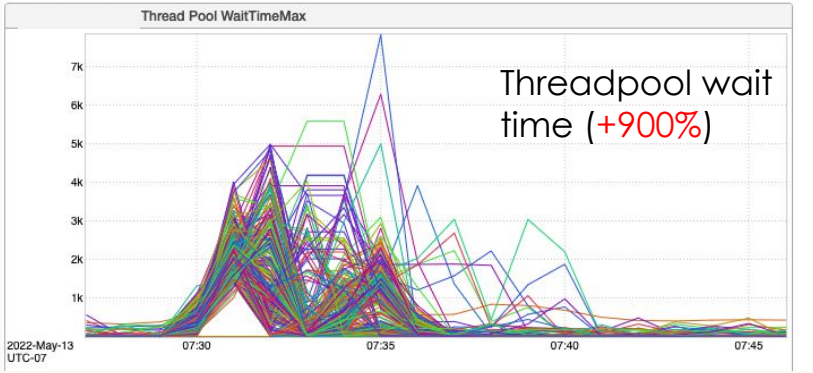
p90 latency (+900%)

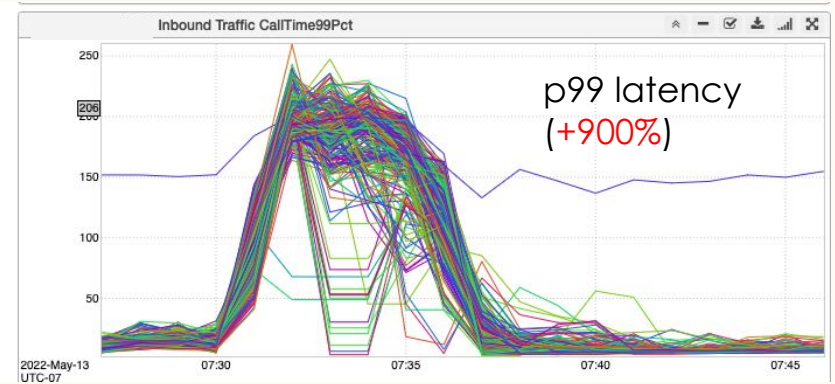
Thread Pool Starvation/Overload detection

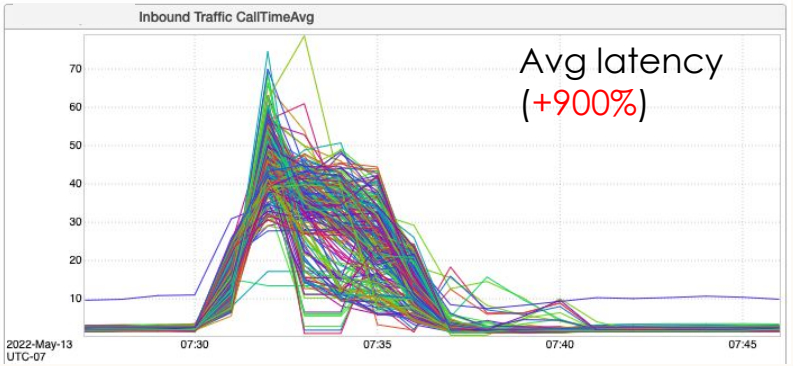
- Objective
  - Early and accurate detection of **Thread Pool Starvation**
- Thread Pool Overload Detection algorithm
  - Monitor ThreadPool Queue wait times
  - If a threshold is consistently breached, flag an Overload
- Rationale
  - HB and GC detect overload of a physical resource. Thread Pool Detector detects overload of a "virtual"/non-tangible resource
  - Helps alleviate backed up downstreams and/or slowed down DBs

Threadpool Overload detection case in prod









Agenda

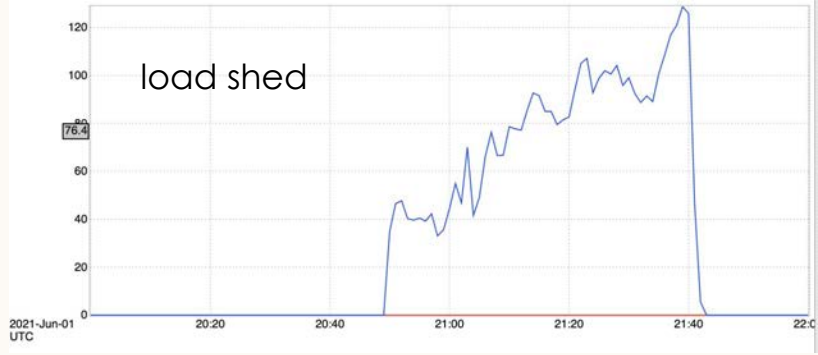
- Hodor Framework Overview
- Overload Detectors: CPU, GC and ThreadPool
- **Overload Remediation: Adaptive load shedder and Request Retries**
- Monitoring and Rollout
- A Success story
- Related/Future work

Adaptive Load Shedder

- Listens to signals from the detectors
- Modulates allowed inbound concurrency
  - Shed aggressively and re-ramp conservatively
- Shed requests by returning HTTP 503 which can be idempotently retried
- Percentage based shedder did not work well during experimentation

--- control host: no load-shedding

--- treatment host: with load-shedding



Overload Failure Retry

- Minimize member impact due to load-shedding
  - Automatically retry requests rejected due to overload
  - Fail quickly and retry safely (avoiding retry storms)
    - Client Side Budget
    - Server Side Budget

Agenda

- Hodor Framework Overview
- Overload Detectors: CPU, GC and ThreadPool
- Overload Remediation: Adaptive load shedder and Request Retries
- **Monitoring and Rollout**
- A Success story
- Related/Future work

Monitoring & Rollout

- When rolling out Hodor we must ensure to not cause unnecessary traffic drops.
- Service analysis and evaluation are a core component of our rollout process.

  - Deploy the overload detectors in monitoring mode to gather data about when they were firing, and if those firings were false positives.

  - Correlate firing data from detectors with performance metrics to determine if there have been any false positives indicating that the service may not be a good candidate for adoption yet.

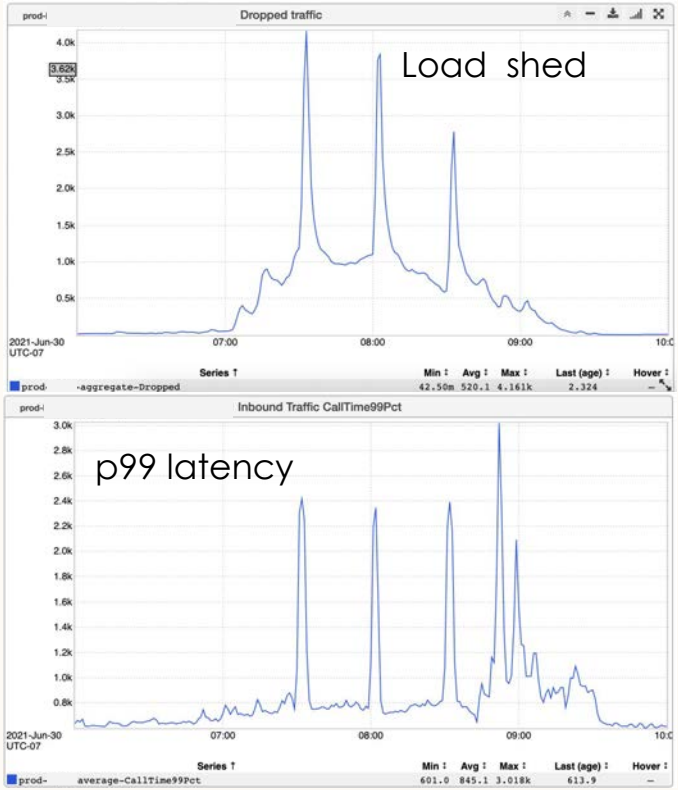
"Adding overload detectors to our services has surfaced unexpected behavior that owners were generally not familiar with."

Agenda

- Hodor Framework Overview
- Overload Detectors: CPU, GC and ThreadPool
- Overload Remediation: Adaptive load shedder and Request Retries
- Monitoring and Rollout
- **A Success story**
- Related/Future work

LinkedIn Flagship app incident mitigated

- Traffic shift led to aggressive load-shedding due to Hodor kicking-in
- Previous traffic shifts did not have a similar response
- Loadshedding likely turned a major member impact incident into a minor one; continued to serve 80% traffic
- Led to quicker discovery of the issue



Agenda

- Hodor Framework Overview
- Overload Detectors: CPU, GC and ThreadPool
- Overload Remediation: Adaptive load shedder and Request Retries
- Monitoring and Rollout
- A Success story
- **Related/Future work**

Related/Future work for Hodor

- Traffic Tiering: Categorize by impact and importance
- New Detectors (Latency based, Eventloop)
- Elastic cluster scaling

Thank you!