Continuous Profiling To Generate Service Performance Insights

Capture code level insights at a time when they matter

LinkedIn

About Me

Saurabh Badhwar

Staff Software Engineer @ LinkedIn, Earlier @ Red Hat

Working on Service Performance & Insights

Authored “Building Enterprise Applications with Python” & “Web Development with Django”

Agenda

1. How We Do Profiling at LinkedIn Currently
2. Our Jump into Continuous Profiling
3. Automated Analysis

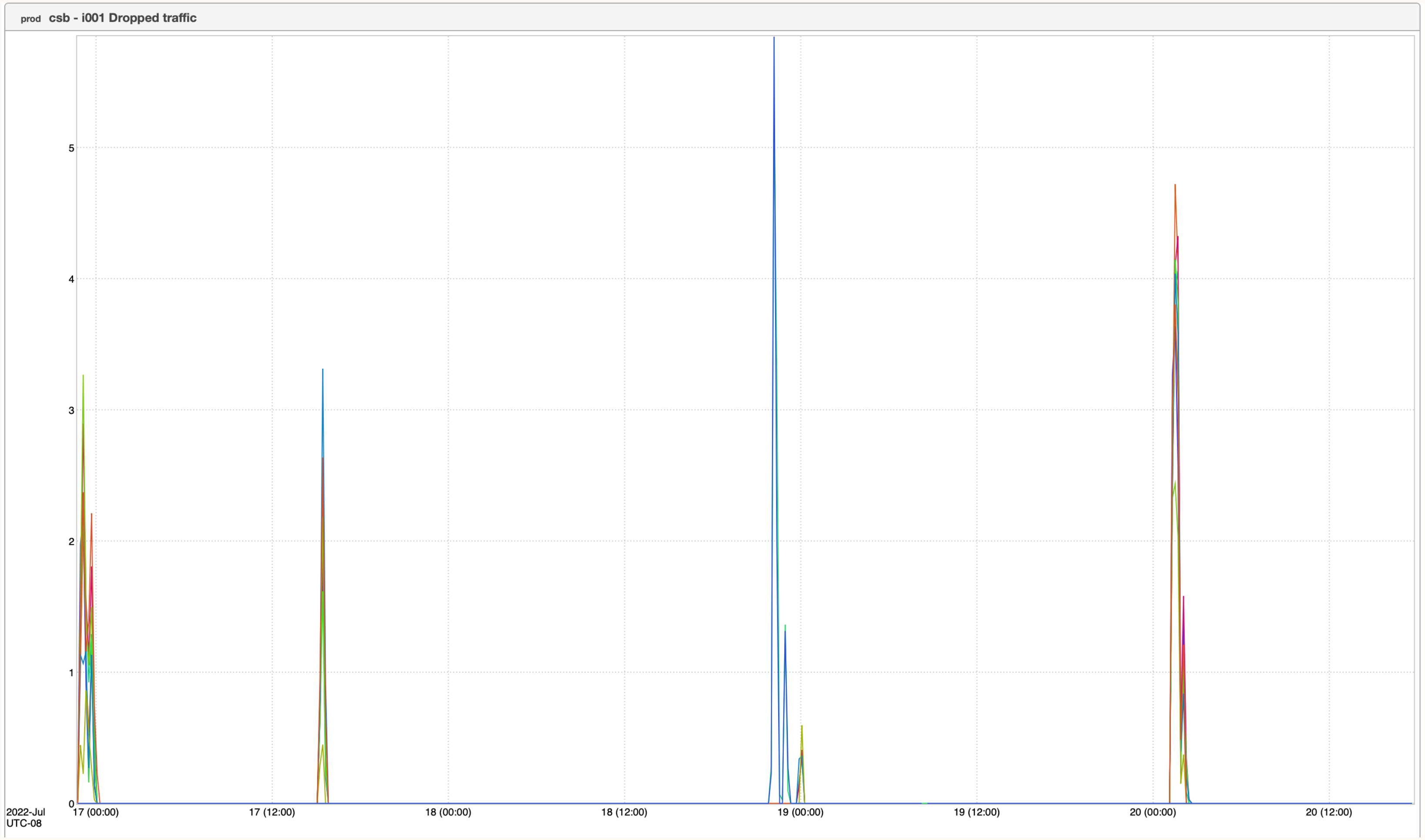
Profiling at LinkedIn

Profiling at LinkedIn

- Centralized Profiling Service – On-Demand Profiler
- 50+ user triggered sessions & 1000+ auto triggered sessions per day
- Profiling is On-Demand in Nature and requires engineering intervention to start
- Results available on a centrally hosted UI to analyze and compare profiling sessions as Flamegraphs
- APIs available for Integration

Issues Don't Have a Predictable Pattern and Engineers are not available every time

Exhibit A: Repeated Traffic Drops by One of the Production Services

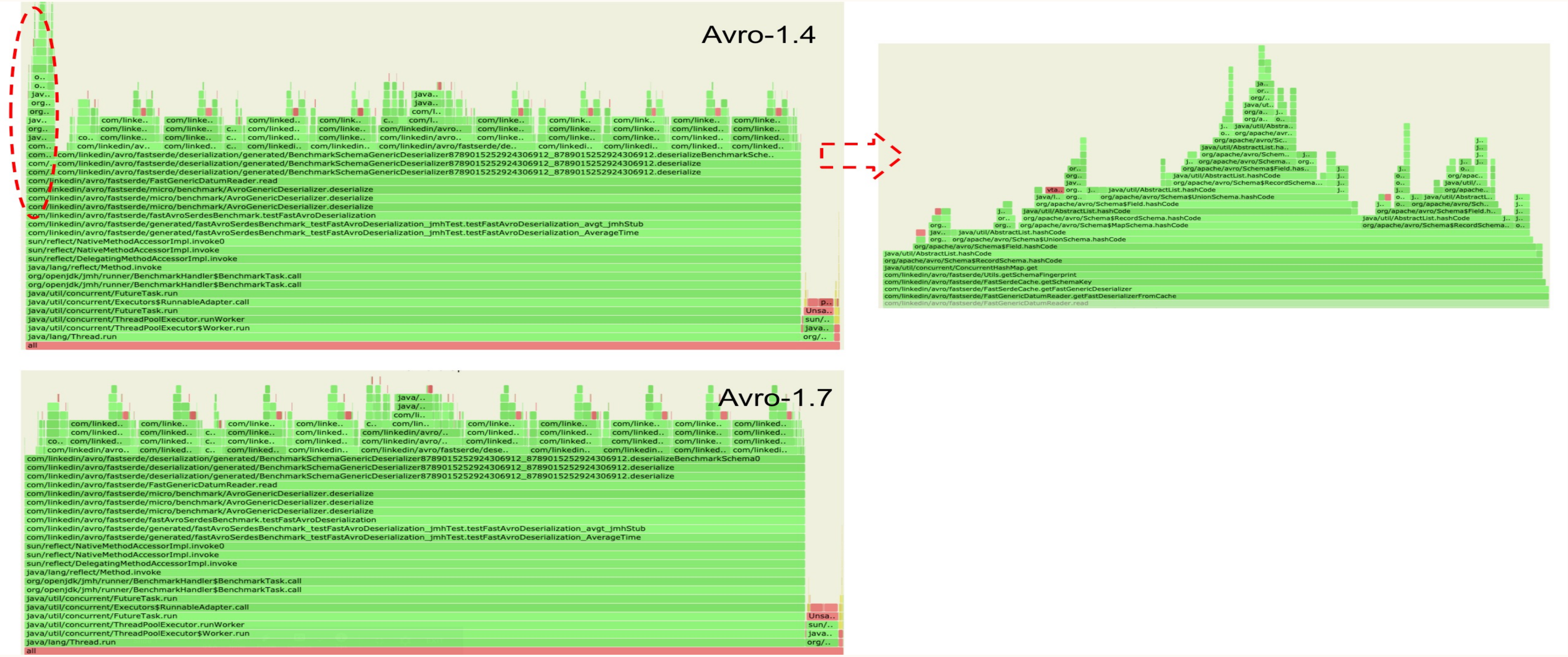


Traffic Drops happen because LinkedIn's [quality of Service detection](#) has detected that service is in a degraded state and may fail unless the traffic is reduced

Exhibit A: Repeated Traffic Drops by One of the Production Services

- Short lived (<20 mins)
- Sporadic in nature
- No specific pattern timings

Exhibit B: LinkedIn's migration to AVRO fast-serdes



Limitations with Current Architecture

- Profiling sessions require engineering intervention and manual triggering
- Profiling during events of interest can require synchronization of timing
- End users may not have Baseline profiles to compare the results with
- Looking for impact across longer time periods is not possible

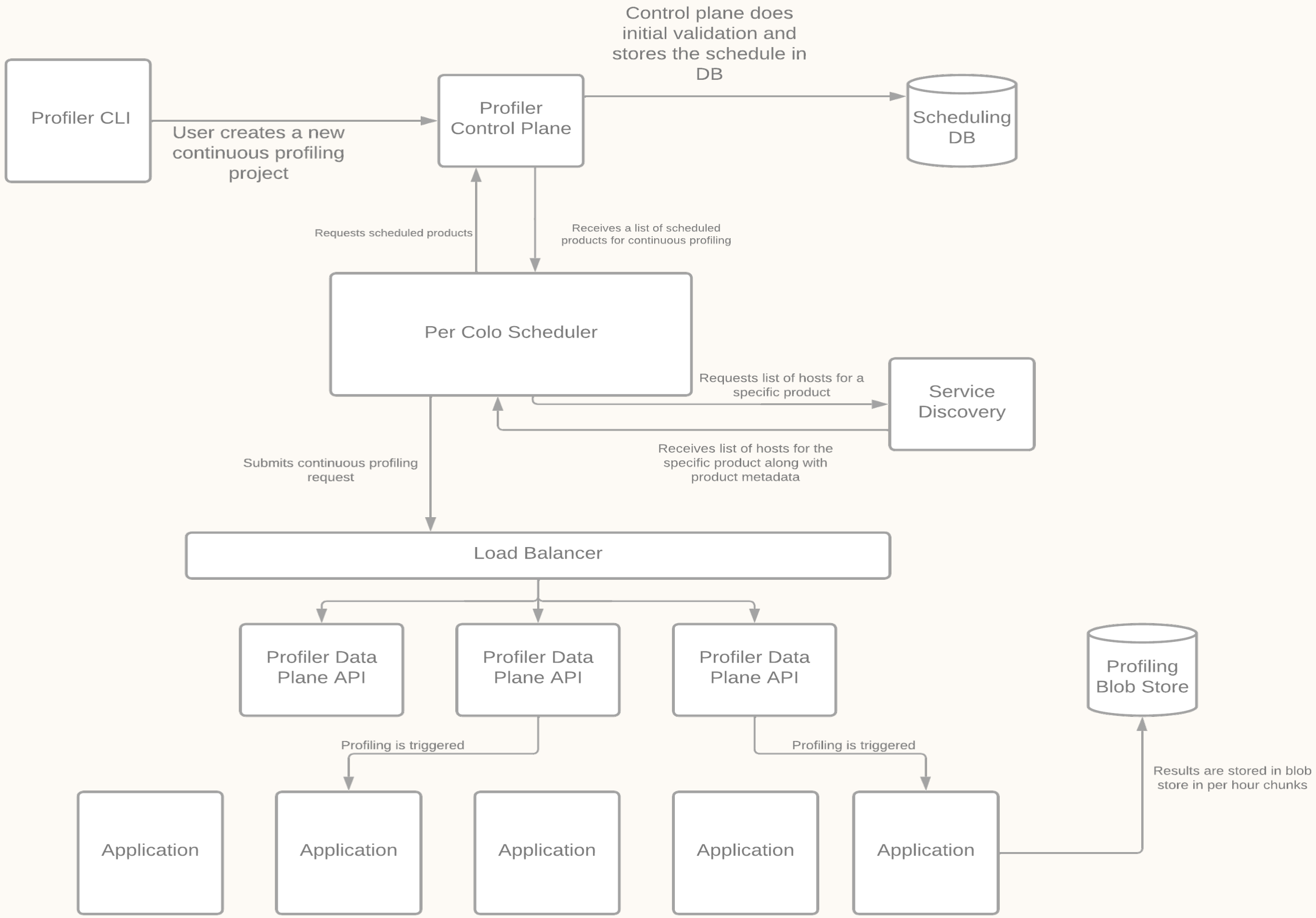
Setting up the base with continuous profiling

24x7 Application Monitoring for Gaining Insights into Application Performance

Continuous Profiling as the base infrastructure

- Applications get profiled 24x7 with a minimal overhead (<0.5%)* and the results get collected continuously
- Ability to do time window-based analysis
- Enabling comparison of profiles across different dimensions
- Enabling automated analysis leveraging the central profiling datastore

Continuous Profiling as the base infrastructure



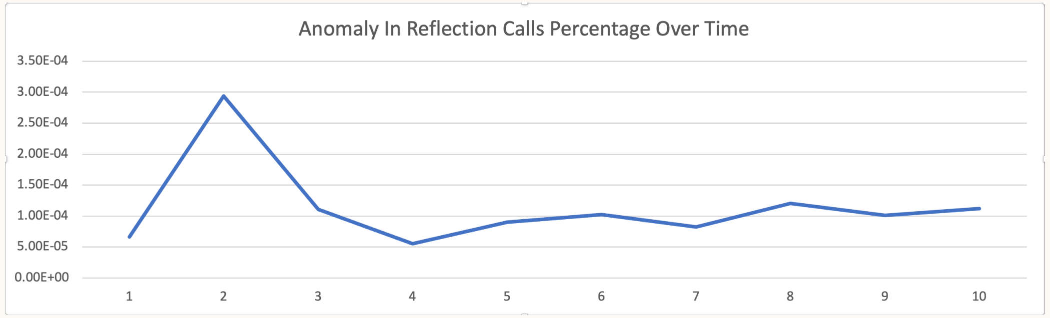
Automated Analysis

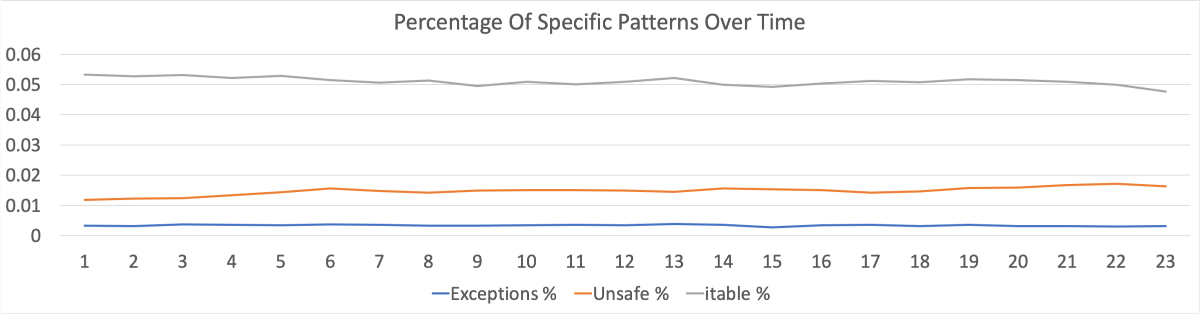
Automated Analysis

- Identifying known performance problems with help of static pattern analysis and reporting
- Calculating infrastructure library costs
- Analyzing changes related to different events (releases, A/B Test Ramps) by measuring changes in distribution of top CPU consumers

Automated Analysis

- Monitor, identify and RCA slow leaks on method level by profiling data, and provide actionable insights for fixing them
- Use data mining techniques to identify trends. I.e., application activity related to global events or daily routine
- Perform anomaly detection on continuous streams of data





Tagging code to specific metrics

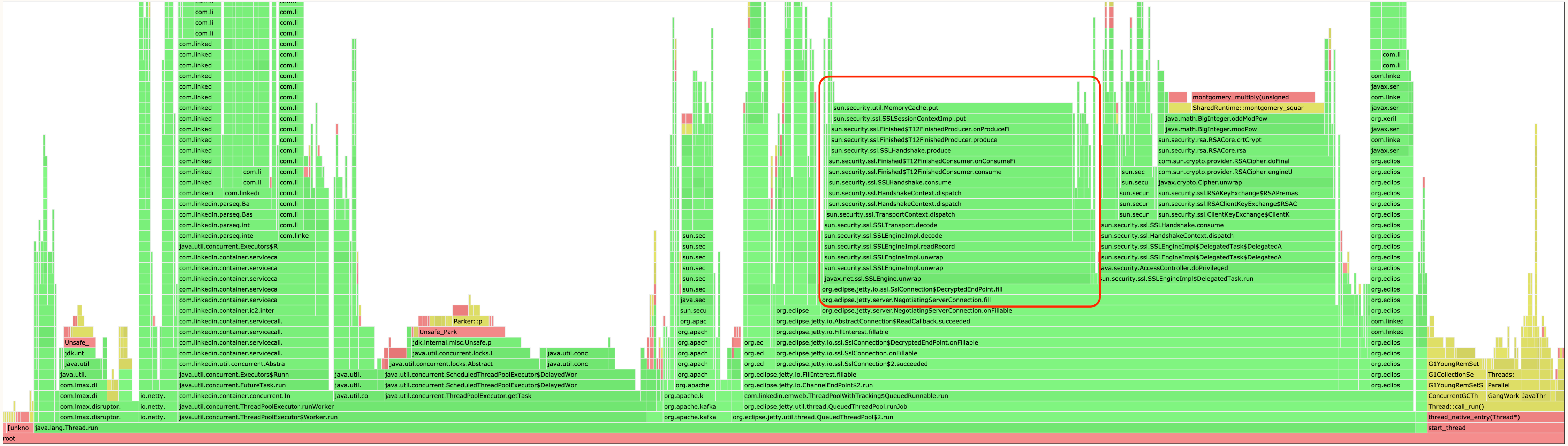
- Consume the raw profiling data in Hadoop/Spark jobs
- Leverage pattern matching for namespaces
  - Example: org.slf4j.logger | org.apache.logging -> Logging
  - Example: com.linkedin.kafka -> Kafka messaging
- Count the CPU sample count and emit it as time series metric

What Metrics We Can Monitor Right Now

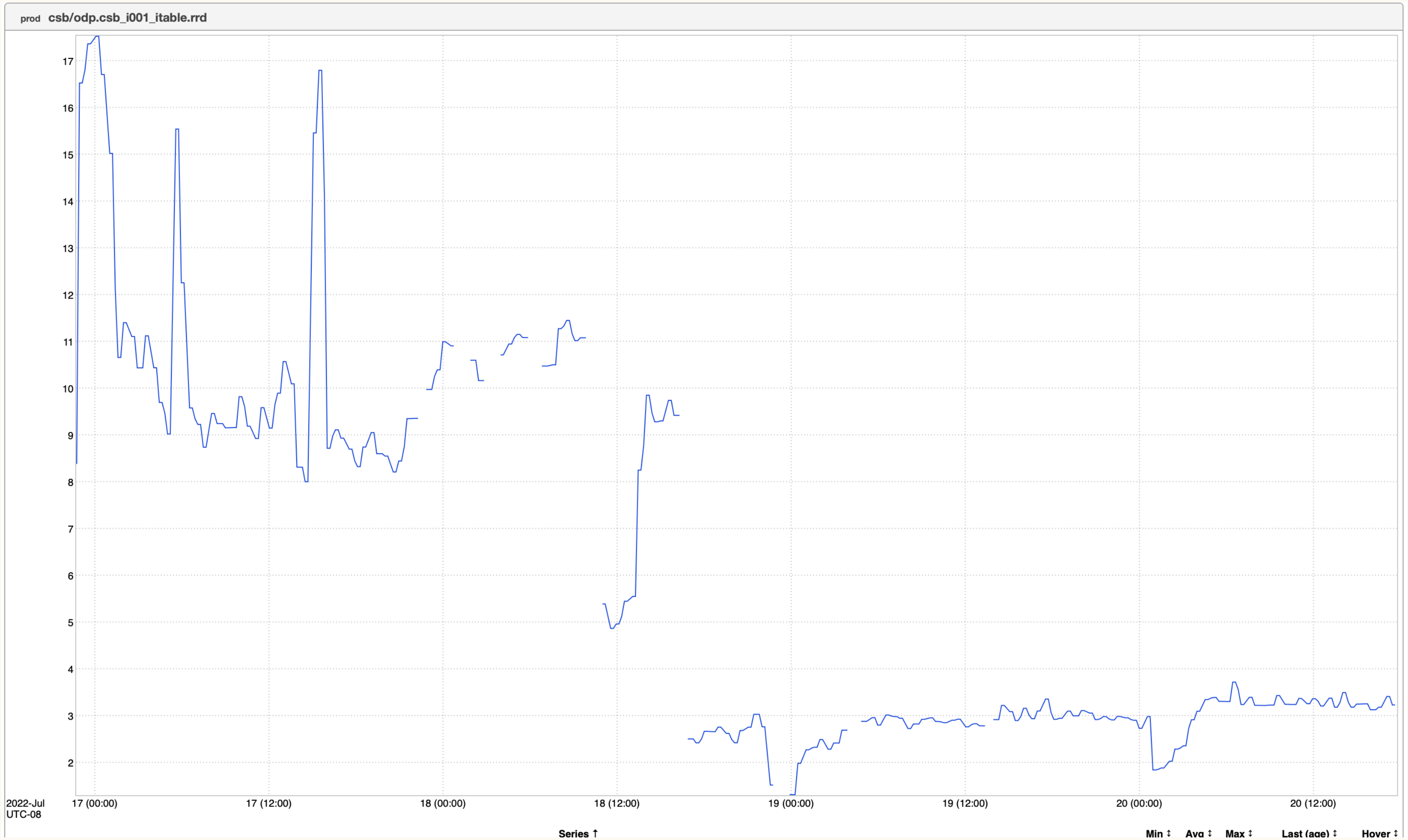
- JVM Internal Metrics – CPU spent resolving Interfaces, CPU spent in reflection calls
- Time spent in frameworks – log4j, netty & jetty server, emitting kafka messages, etc.
    - Logging
    - Traffic and request routing: netty & jetty
    - Message emission / consumption: kafka
- Time spent in application logic

Automated Analysis

- Automated bottleneck detection. Issues like JDK-8259886 could be detected and reported automatically.



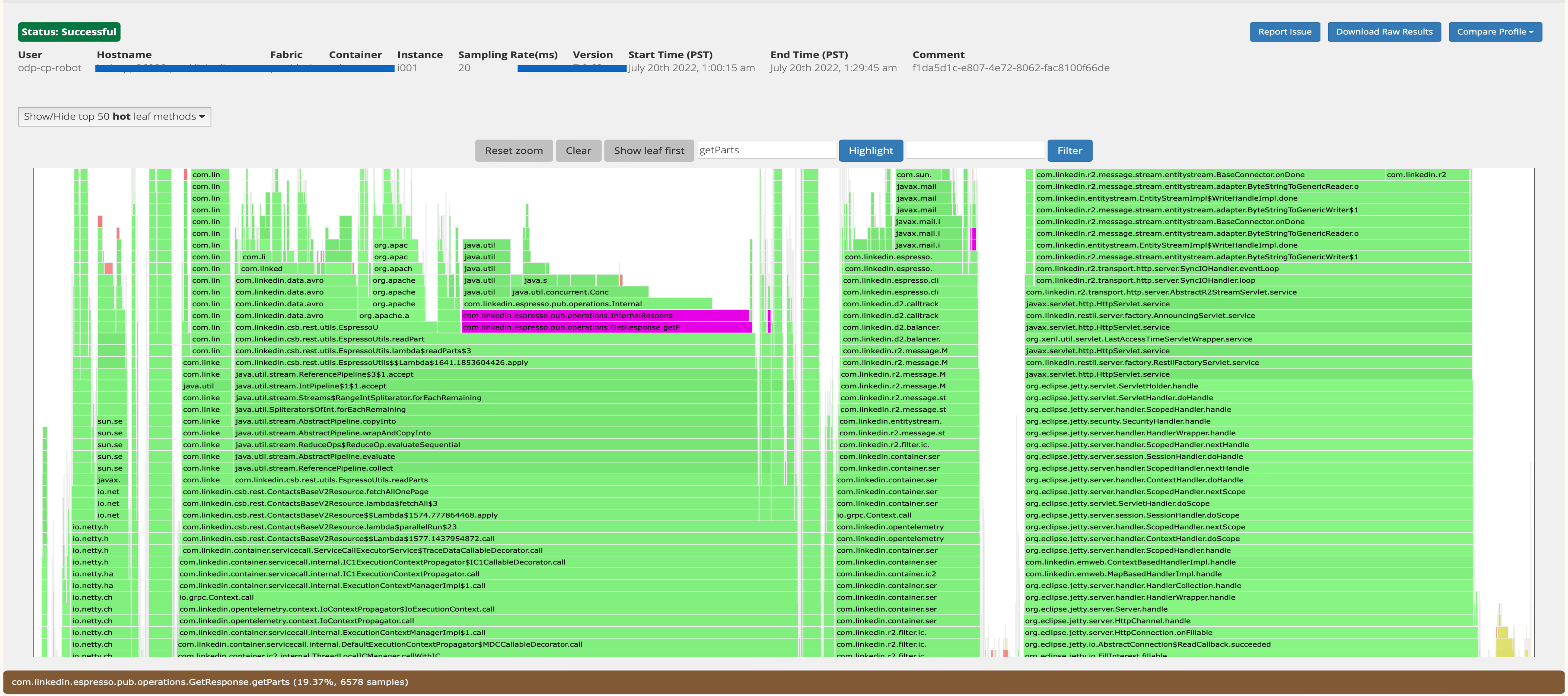
Relooking at our previous issue: Exhibit A



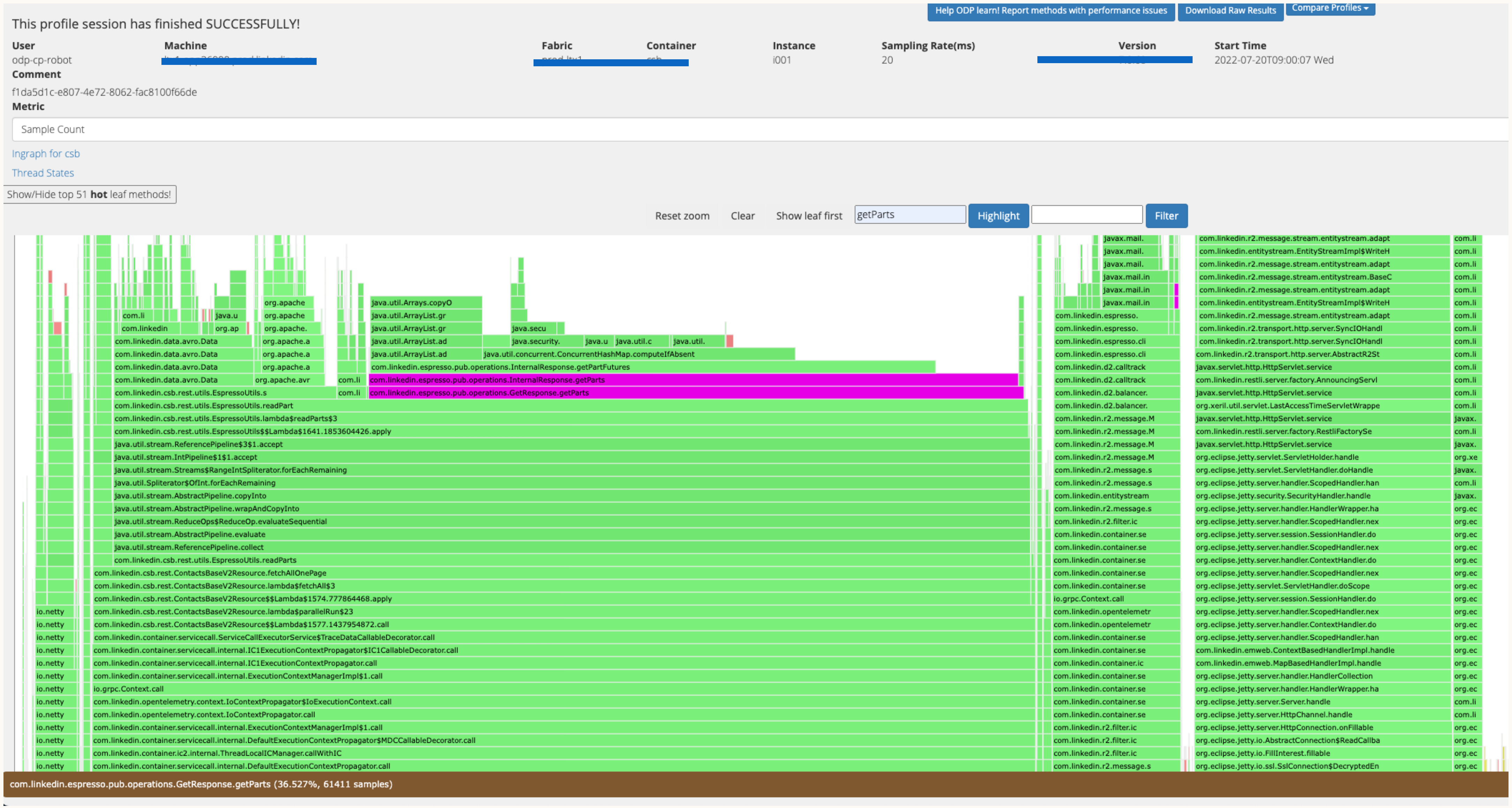
CPU Samples for itables

Relooking at our previous issue: During Overload

Relooking at our previous issue: Before Overload



Relooking at our previous issue: After Fixes (During Overload)



Challenges with continuous profiling

- Application fleet is not homogeneous
- Containerized architecture and multiple deployments a day – Apps can get restarted anytime
- Near-realtime / short lived jobs may not have long enough durations to successfully complete profiling
- On-boarding every instance for every service = massive data storage per day

Problem with deployment homogeneity

- Deployment hardware can be different
- Two deployments could have different service configurations
- Comparing different configurations can mess up continuous profiling data

Solving for deployment homogeneity

- Fetch the similar kind of deployments using the deployment artifactory
- Select similar configurations from service config tags
- Match the hardware configuration while generating insights

The Ever Growing Storage Needs

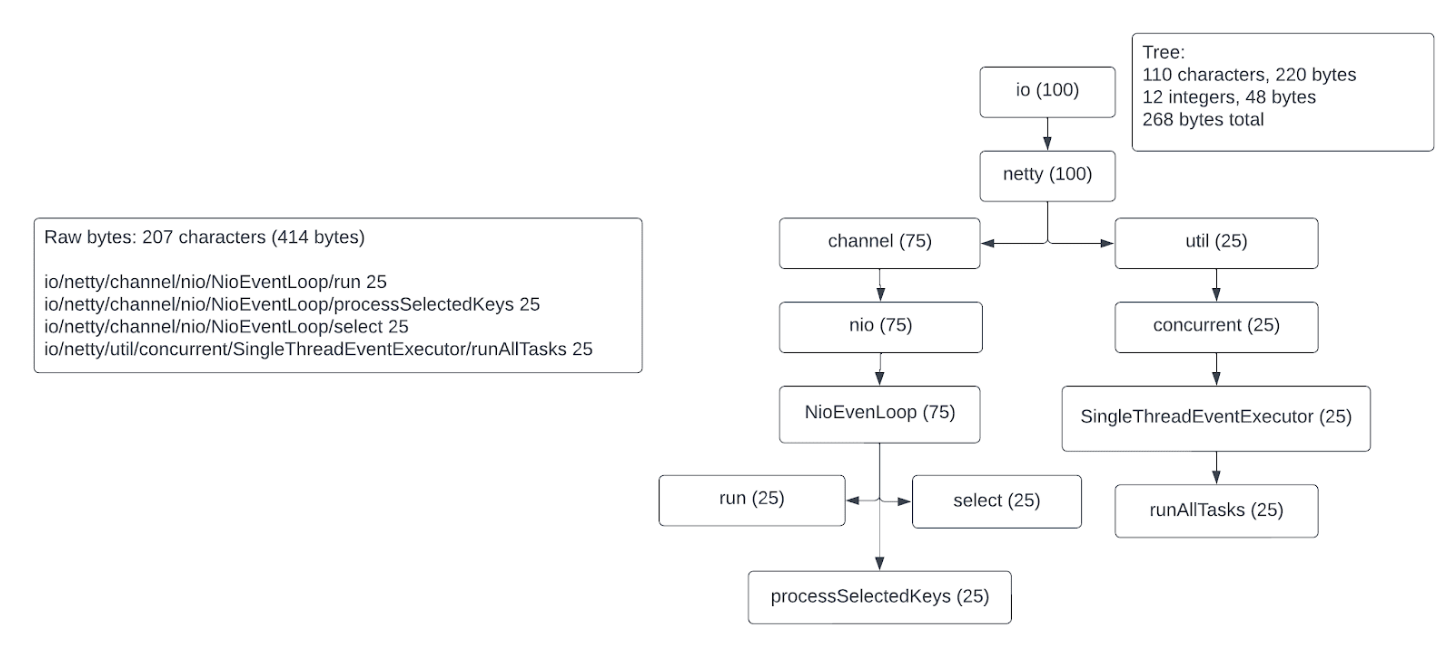
- ~2k production services
- Average fleet size of 30 nodes
- Average per profile data size: 400 MB
- 2 sessions (each 30 mins long) per hour
- Expected daily storage need ~48 TB

The Ever Growing Storage Needs: Solving for challenges

- Profile only two hosts per unique dimension pair (dimension = data center, config, app version)
- Use compression to reduce data size for storage
- Set data retainment policy for blob storage aggressively
- Leverage cheaper long term storage options – HDFS
- Focus on insights rather than retaining raw data for longer periods

The Ever Growing Storage Needs: Steps Ahead

- Opportunity to optimize the data sizes further
- Majority of the functions stay the same over a long term period
- We can trade off some CPU for increased compression rates



What Makes Continuous Profiling Possible for Us

- Async-profiler
- Python and Py-spy
- Linux perf

The Journey Ahead

- Make the insights available to better understand overloads
- Detect common issues impacting majority of the production services at LinkedIn
- Combine with tracing data to provide a holistic experience while performing RCA

Thank you

Find Me On:

LinkedIn: https://www.linkedin.com/in/sbadhwar/

Twitter: @saurabh_badhwar

Blog: https://www.saurabhbadhwar.xyz/blog

Q&A