MicroView: Cloud-Native Observability with Temporal Precision

Alessandro Cornacchia
Politecnico di Torino

Theophilus A. Benson
Carniege Mellon University

Muhammad Bilal
Unbabel

Marco Canini
KAUST

ABSTRACT

We present MicroView, a system designed to improve the accuracy and timeliness of observability in cloud-native applications, while minimizing overhead. MicroView stands out from conventional observability tools by incorporating metrics processing stages at every node within a local lightweight data-plane. We preliminary demonstrate its benefits for distributed tracing and outline a set of architectural choices focused on offloading the MicroView data-plane to IPU accelerators, such as a BlueField-3 SmartNIC, thus limiting the interference with running services.

ACM Reference Format:
Alessandro Cornacchia, Theophilus A. Benson, Muhammad Bilal, and Marco Canini. 2023. MicroView: Cloud-Native Observability with Temporal Precision. In *Proceedings of the CoNEXT Student Workshop 2023 (CoNEXT-SW '23), December 8, 2023, Paris, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3630202.3630233

1 INTRODUCTION

Microservice observability is a key requirement for troubleshooting cloud-native applications, as it provides visiblity about their internal state. Observability tools collect a wealth of monitoring data — i.e., metrics, request traces and logs — which is then used to detect and diagnose failures and identify performance bottlenecks.

Unfortunately, observability can create a significant overhead on server resources thus creating resource contention and interference with user services. This overhead is mainly generated by data copies and network stack processing [4] to communicate with the monitor backend. Even worse, it grows with the scale of monitored components and the frequency at which data is collected. In practice, operators need to resort to relaxed sampling rates for data collection, which sacrifices the quality of observability itself, such as accuracy and timeliness. Although scheduling ad-hoc CPU bonding for the observability processes or vertically scaling the infrastructure would mitigate the problem, these solutions are neither energy-efficient nor cost-efficient.

This work proposes a system to guarantee accuracy and timeliness of observability, while limiting the overhead and interference with running applications. Our design hinges on the observation that for metrics, the overhead is dominated by the *ingestion* costs rather than *generation* costs (Table 1). This allows us to — relatively cheaply — increase the temporal granularity at which new metrics samples are produced, and to decouple local generation rate from ingestion rate. In light of these observations, we address the benefits and challenges of *(1)* running a metrics processor on each node that can handle high metrics generation rate, and *(2)* exploiting such intelligence to locally extract actionable signals and assist performance debugging tasks. We take distributed tracing as a showcase example and demonstrate that MicroView can improve the coverage of anomalous requests (i.e., whose latency violates SLO) by approximately 5× compared to head-based sampling. Finally, we delineate architectural choices to offload the metrics processor to emerging Infrastructure Processing Units (IPUs) [5] accelerators, such as BlueField-3 SmartNIC.

CoNEXT-SW '23, December 8, 2023, Paris, France

ACM ISBN 979-8-4007-0452-9/23/12.
https://doi.org/10.1145/3630202.3630233

2 PROPOSED DESIGN

2.1 MicroView overview

The proposed system architecture is shown in Figure 1. Our system differs from traditional observability architectures [4] in that it adds metrics processor locally at each node. The ***data-plane*** is where metrics processing takes place. It consists of a bank of per-microservice classifiers. Each classifier periodically is fed with a metric vector relative to its corresponding microservice. It applies pre-processing transformations (e.g., accumulation, data whitening, etc.), and decides if the vector sample is anomalous or not. We adopted a streaming sketch-based binary classifier [2], as it operates in single-pass without requiring local storage of samples, yet supporting continual-learning from new samples. At its heart, the classifier learns a low-dimensional reconstruction basis for the data (i.e., sketch), and detects anomalies if it cannot reconstruct an input sample within a predefined error tolerance. The ***control-plane*** consists of MicroView agents that orchestrate the interaction between the metrics sources (e.g., microservices) and the data-plane. If the data-plane sits on an IPU accelerator, MicroView agents ensure the IPU can access the memory regions where the metrics variables reside. Additionally, it produces actionable alerts to observability

Metrics observability configuration		**CPU usage at the monitored node**
Generation rate [Hz]	*Ingestion rate [Hz]*	
1	1	12-13%
1	1/30	3-4%
1/30	1/30	2%

Table 1: CPU consumption of a single monitored Kubernetes node, running the `Online-boutique` workload [1]. Ingestion rate is the frequency at which a Prometheus monitor — on a different node — queries metrics from the pods. Generation rate refers to the frequency at which metrics are created/updated within the pods. We tuned the generation rate by changing the `housekeeping_interval` parameter in cAdvisor.

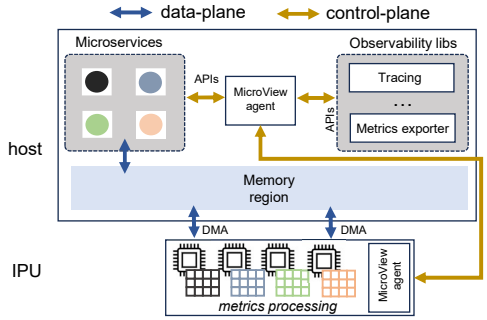

Figure 1: MicroView architecture. Unlike traditional observability architectures, we propose to add local metrics processing stages at each node.

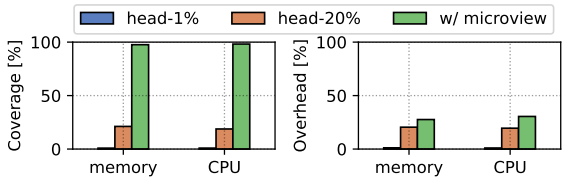

Figure 2: Performance-overhead trade-off of sketch-assisted distributed tracing for different anomalies.

libraries based on sketch classifications. The specific action depends on the desired use-case, an example is provided in the next section.

2.2 Use-case: distributed tracing

Metrics reveal the internal state of applications and containers: their unexpected variation potentially indicates an anomalous state, which negatively impacts user requests. Online metrics analysis can anticipate anomalous executions, thus enabling distributed tracing to sample [6] informative requests. To achieve its goal, the MicroView agent combines outputs from different classifiers. If at least one classifier detects an anomalous state, the agent triggers the distributed tracer. At this point, the tracer samples all incoming requests until the next classification cycle. In this regard, MicroView substantially differs from existing tools, which collect metrics and traces independently and correlate them offline. Notably, MicroView is complementary to state-of-the-art Hindsights' retroactive sampling [6].

3 PRELIMINARY EVALUATION

Experimental setup. We implemented a preliminary prototype in Python that processes datasets of metrics and traces offline. The datasets are collected from running a widely used benchmark application [1] on a 4-node Kubernetes (v1.25.5) cluster, each equipped with 8 Intel Xeon E3-1230v6 CPU cores at 3.50GHz, 32 GB of RAM, and running Ubuntu 22.04. We deployed the Istio service mesh on the cluster and used Locust for load generation. We instrumented for observability with a Jaeger tracer and a Prometheus instance that collects service-level and container-level metrics every second. Service-level metrics include Istio metrics — such as `istio_requests_total` — and custom application metrics whenever available (e.g., Redis), while container-level metrics are resource usage counters (CPU, memory, disk I/O, etc.) exported by cAdvisor. We fit the sketch classifiers with an initial training phase. During this phase, we also tune hyperparameters for each sketch (i.e., microservice) separately. First, we use a *healthy* metrics dataset from which initialize the sketches to learn "normal" behavior [2]. We obtain the dataset when the application runs in underload conditions and with over-provisioned vCPU and memory limits. Second, we inject faults in the service and produce a second dataset, that we use to pick the hyperparameters that give the best F1-score. We use ChaosMesh for fault injection and simulate stress scenarios on the containers, such as CPU and memory.

Can MicroView help tracing? We use the pre-trained sketches to guide trace sampling (as per Sec.2.2). Every 30 seconds we inject short failures lasting 5 seconds on a randomly selected service. We compare against head-based sampling in terms of coverage, i.e., percentage of collected anomalous traces, and overhead, measured as percentage of false positives. A trace is defined as anomalous when exhibits high latency or contains error codes. Figure 2 shows that tracing + MicroView achieves nearly total coverage, 5x better than sampling 20% of the request. For more than 70% performance boost, MicroView adds 5% overhead. This is because head sampling relies on luck to capture faulty traces, while MicroView is guided by metric signals.

4 RESEARCH AGENDA

We delineate challenges and research avenues left as a future work.

Host-IPU communication. The data-plane, that we plan to implement on a IPU accelerator, needs to access metrics that sit on the host OS [5]. A natural choice is to use DMA technology and bypass the host CPU. In this space, we identified two competing alternatives: RDMA and NVIDIA DOCA libraries, which we plan to compare. Complementary to it, the next step on the host side is the definition and evaluation of the interfaces between MicroView and the microservices for metrics creation and update.

Alternative use-cases. Different metrics have different natures. Some fluctuate on short time scales (e.g., CPU, power), while other stays constant and change only in response to human reconfigurations. MicroView can work as a filter for metric ingestion, dynamically deciding which metrics are worth ingesting, which can save storage costs for the tenant [3] and bandwidth for the provider.

REFERENCES

[1] 2023. Online-boutique. https://github.com/GoogleCloudPlatform/microservices-demo
[2] Hao Huang and Shiva Prasad Kasiviswanathan. 2015. Streaming anomaly detection using randomized matrix sketching. *VLDB Endowment* 9, 3 (2015), 192–203.
[3] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. 2017. Sieve: Actionable insights from monitored metrics in distributed systems. In *ACM/IFIP/USENIX Middleware*.
[4] Zhe Wang, Teng Ma, Linghe Kong, Zhenzao Wen, Jingxuan Li, Zhuo Song, Yang Lu, Guihai Chen, and Wei Cao. 2022. Zero Overhead Monitoring for Cloud-native Infrastructure using RDMA. In *USENIX ATC*.
[5] Xingda Wei, Rongxin Cheng, Yuhan Yang, Rong Chen, and Haibo Chen. 2023. Characterizing Off-path SmartNIC for Accelerating Distributed Systems. In *USENIX OSDI*.
[6] Lei Zhang, Zhiqiang Xie, Vaastav Anand, Ymir Vigfusson, and Jonathan Mace. 2023. The Benefit of Hindsight: Tracing Edge-Cases in Distributed Systems. In *USENIX NSDI*.