Scaling OpenTelemetry on GKE Autopilot: Throughput, Backpressure, and Latency in a Real Cloud Deployment

Omar Bin Kasim Bhuian
SysCore Lab,
Sejong University
Seoul, South Korea
omarbinkasimsefat@gmail.com

Sang-Hoon Choi
SysCore Lab,
Sejong University
Seoul, South Korea
csh0052@gmail.com

Young-Soo Kim
Electronics and Telecommunications Research Institute,
Seoul, South Korea
blitzkrieg@etri.re.kr

Hongri Liu
Weihai Cyberguard Technologies Co., Ltd., Weihai 264209, China
liuhr@cyberguard.com.cn

Ki-Woong Park
Department of Information Security and Convergence Engineering for Intelligent Drone, Sejong University
Seoul 05006, South Korea
woongbak@sejong.ac.kr

***Abstract*— In this study we evaluate a production-style OpenTelemetry (OTel) pipeline on Google Kubernetes Engine (GKE) Autopilot under a sustained trace workload, instrumenting end-to-end ingestion→processing→export and scraping Collector self-telemetry and spanmetrics with Prometheus over a 30-minute run. The Collector averaged 573 spans/s accepted and 561 spans/s exported, yielding 97.84% within-window export efficiency and peaking near 924 spans/s. A brief saturation interval (150 s) produced a queue peak of 6.38, short drops (peak 3.49 spans/s, ~358 total), and p95 inflation to 116 ms; median latency remained low (p50 ≈ 12.9 ms) and recovered after pressure subsided. Resource footprint was modest (≈0.36 CPU cores, 0.21 GiB memory, sub-Mbps network), indicating headroom. We document integration pitfalls (OTLP endpoint/protocol) and show that queue growth and exporter errors anticipate p95 tails. The study contributes reproducible methodology, quantitative evidence of cloud-scale OTel scalability, and operator guidance for capacity planning and alert design.**

Keywords*— *OpenTelemetry (OTel), Distributed tracing, Cloud observability, Kubernetes (GKE Autopilot)

I. Introduction and Background

Modern microservice platforms require high-fidelity observability to localize latency regressions and make defensible capacity decisions. OpenTelemetry (OTel) has become the vendor-neutral standard for generating and transporting traces, metrics, and logs via stable APIs/SDKs and the OpenTelemetry Protocol (OTLP). The OTel Collector implements configurable "receive → process → export" pipelines that decouple application instrumentation from the export path, enabling multi-backend delivery without per-agent lock-in[1].

Operational metrics in this study are scraped with Prometheus, who's dimensional, label-rich time-series model and PromQL make it a de-facto substrate for aggregation and alerting in cloud-native systems[2]. We deploy in Google Kubernetes Engine (GKE) Autopilot, a managed mode in which Google provisions and scales the node layer and applies opinionated hardening—an environment representative of production clusters where teams focus on workload configuration rather than infrastructure plumbing[3].

At scale, tail latency rather than the mean governs user experience: delays that are rare at small scale can dominate perceived performance as fan-out grows. Dean and Barroso formalized this "tail at scale" effect and argued for tail-tolerant techniques that manage variability before it breaches service SLOs. This motivates our emphasis on p50/p95 behavior alongside throughput and backpressure signals[4].

Beyond foundational systems, a growing body of work studies how to extract actionable insights or reduce the cost of telemetry at scale. Thalheim et al.[5] propose Sieve, a framework that reduces monitored metrics by 10–100× and infers dependencies between microservices using Granger causality, enabling autoscaling and root-cause analysis while significantly lowering CPU, storage, and network overhead. Zhang et al. introduce Hindsight, a retroactive sampling system that records detailed trace data locally and only persists it when symptomatic edge cases (e.g., high tail latency, errors, bottlenecked queues) are detected, thereby scaling to millions of requests per second with nanosecond-level overhead[6].

Building on this context, Omar examined OTel in edge/hybrid environments and reported integration sensitivities that affect end-to-end observability (e.g., protocol/endpoint configuration and resource constraints). The present study complements that line of inquiry by profiling a cloud-hosted OTel pipeline on GKE Autopilot under sustained load, with a focus on how Collector telemetry (throughput, queues, drops, exporter errors) relates to tail latency in practice[7].

Finally, our framing aligns with upstream guidance: the OTel specification and Collector architecture describe stable

signals (traces/metrics/logs), pipeline components, and OTLP transport semantics that we rely on to structure measurements and interpret backpressure. Using Prometheus' scrape model keeps the analysis reproducible and independent of any proprietary backend[8].

Beyond evaluating raw scalability, this study contributes to a broader research agenda on cloud observability: whether managed Kubernetes environments introduce unique constraints on telemetry pipelines, and how Collector-level signals (queue growth, exporter errors) can be used to anticipate tail latency before it breaches SLO thresholds. By situating our work within prior research on tracing system scalability and "tail at scale" effects, we show how OTel behaves in a real-cloud environment and how our findings complement and extend previous approaches.

II. System and Methodology

We deploy a production-style OpenTelemetry (OTel) pipeline on Google Kubernetes Engine (GKE) Autopilot in the asia-northeast3 region. The system consists of an OTel Collector configured for OTLP over HTTP on port 4318, Prometheus scraping the Collector's self-telemetry and spanmetrics, and a sustained trace generator (telemetrygen/tracegen). The observation window is 30 minutes sampled every 15 seconds, with timestamps in UTC.

Prometheus scrapes a fixed set of signals: accepted and exported spans per second, p50 and p95 latency (seconds), the Collector's internal queue size, dropped spans per second, exporter errors per second, and resource metrics (CPU cores, memory bytes, and network RX/TX bytes per second). Raw time series (CSV) and figures (PNG/PDF) are exported for offline analysis and reproducibility.

The analysis focuses on throughput, tail latency, and backpressure using simple, auditable computations from the time series. For throughput, we summarize means, peaks, and totals over the window and define export efficiency over the same window $\mathcal{T}$ as the ratio of exported to accepted trace volume, computed directly from the sampled rates:

$$\eta_{\text{exp}} = \frac{\sum_{t\in\mathcal{T}} \lambda_{\text{exp}}(t)}{\sum_{t\in\mathcal{T}} \lambda_{\text{acc}}(t)}, \qquad (1)$$

where $\lambda_{\text{acc}}(t)$ and $\lambda_{\text{exp}}(t)$ denote accepted and exported spans per second at time t (15-second cadence). To characterize backpressure, we use the Collector's queue size $Q(t)$ and mark saturation when $Q(t) > 3$; the corresponding duration and temporal co-movement with p95 latency are reported in the Results section. We also examine short lead–lag relationships between $Q(t)$ and p95 to assess whether queue growth precedes tail inflation. Endpoint/protocol configuration (OTLP/HTTP on :4318) and the scrape cadence are fixed for all runs to ensure comparability and reproducibility.

III. Results

A. Throughput and Export Efficiency

Over the 30-minute window (UTC 09:25:48–09:55:48; Δt=15 s, N=121), the Collector admitted a mean of 573.2 spans/**s** and exported 560.8 spans/**s**; both streams reached a common peak of 923.7 spans/**s** during the step load. The

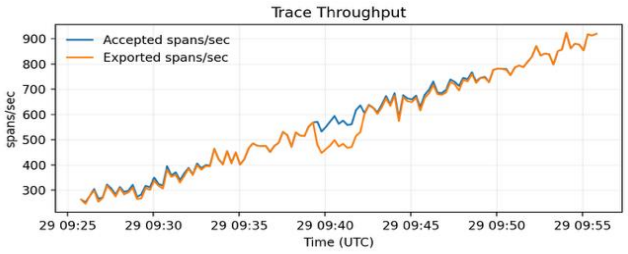

Fig. 1. Trace Throughput (Accepted vs, Exported)

within-window export efficiency defined in Eq. (1) is 97.84%, and a five-minute moving average of pointwise efficiency maintained a high level (mean 97.53%) with a brief minimum of 90.63% during the ramp. Volume accounting shows a small accepted–exported gap consistent with window boundary and scrape alignment effects; explicit drop counters indicate only short-lived loss. Exported throughput tracks accepted throughput closely outside the short saturation interval and return to baseline without drift.

Accepted and exported spans per second for the OpenTelemetry Collector on GKE Autopilot over the 30-minute window (UTC; 15-s cadence). Exported throughput closely tracks accepted throughput except during a brief saturation interval; within-window export efficiency is computed by Eq. (1).

B. Latency and Tail Behavior

The latency profile derived from spanmetrics shows stable central tendency and transient tails. Median latency p50 averages 0.0129 **s** (peak 0.0255 **s**), while p95 averages 0.0513 **s** with a short excursion to 0.1157 **s** during the load ramp, after which it returns to baseline. Tail dynamics co-vary with backpressure: the Pearson correlation between queue size and p95 at the same timestamp is r = 0.554; allowing a +15 s shift (queue leading p95 by one step) increases the correlation to r = 0.575. A descriptive linear model fitted to the samples,

$$\pi 95_t = \alpha + \beta\, Q_t + \gamma\, \varepsilon\rho\rho\rho\sigma_t + \delta\, \lambda_{\alpha\chi\chi,t} + \epsilon_t,$$

yields $R^2 \approx 0.31$ with $\beta \approx 5.65\times10^{-3}$ **s** per queue unit (≈5.7 ms higher p95 per additional unit of Q), summarizing the association between backpressure and tail inflation (not a causal claim).

End-to-end latency percentiles from spanmetrics across the same window. Median latency (p50) remains stable, while p95 exhibits a short excursion coincident with backpressure; tails recover once the queue drains.

C. Ackpressure and Drops

The Collector's internal queue remains low for most of the run (median 0.28) but peaks at 6.38 during a short saturated period. Using the threshold $Q(t) > 3$, the detected saturation interval spans 150 **s** (09:39:33–09:42:03 UTC). During this interval, explicit dropped spans/s reach 3.49, and integrating dropped_spans_per_sec over the full window yields ~358 spans. Exporter errors occur at a low background rate (mean 0.021/**s**, peak 0.8/**s**) and align temporally with the saturation window. Outside this period, queue size returns to baseline and drops cease, indicating rapid recovery of the pipeline.

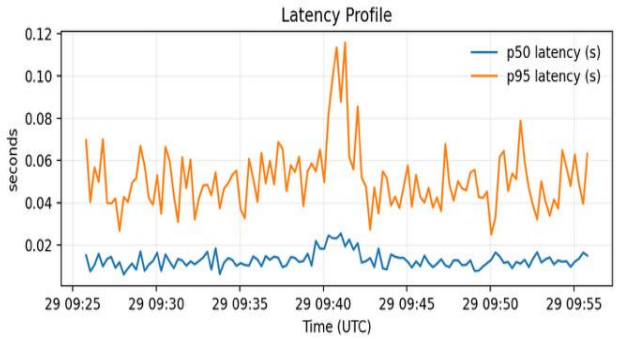

Fig. 2. Latency Profile (p50 and p95).

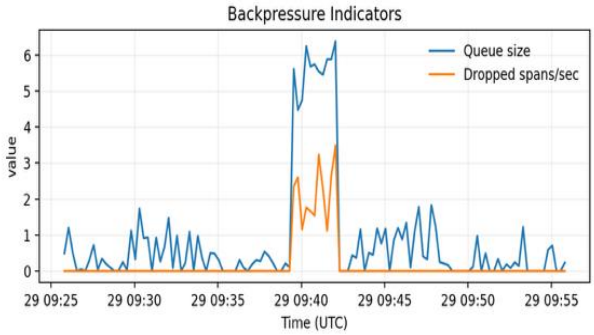

Fig. 3. Backpressure Indicators (Queue Size and Dropped Spans)

Collector internal queue size and dropped spans per second. A short saturation period ($Q(t) > 3$) produces a queue peak and brief drops that align temporally with the p95 spike in Figure 2; exporter errors (not shown) co-occur in the same interval.

TABLE I. *Environment and Headline KPIs*

Category	Value
Cluster / Region	GKE Autopilot / asia-northeast3
Window (UTC)	09:25:48–09:55:48 (Δt = 15 s; 121 samples)
Accepted / Exported (mean)	573.2 / 560.8 spans/s
Accepted / Exported (peak)	923.7 / 923.7 spans/s
Export efficiency η_{exp} (Eq. (1))	97.84%
Totals over window (summary CSV)	1,031,493 accepted / 1,008,983 exported spans
Drops (integrated / peak)	~358 spans / 3.49 spans/s
Queue size (median / peak)	0.28 / 6.38
Saturation interval ($Q > 3$)	09:39:33–09:42:03 UTC (150 s)
p50 latency (mean / peak)	0.0129 s / 0.0255 s
p95 latency (mean / peak)	0.0513 s / 0.1157 s
CPU (mean / peak)	0.36 / 0.58 cores
Memory (mean / peak)	0.21 / 0.22 GiB
Network (mean RX / TX)	0.147 / 0.292 Mb/s
Exporter errors (mean / peak)	0.021/s / 0.8/s

D. *Resource Footprint*

Mean/peak CPU are 0.36/0.58 cores, memory 0.21/0.22 GiB, and average network 0.147/0.292 Mb/**s** (RX/TX). These values indicate headroom under the tested load and are summarized with other KPIs in Table 1.

Cluster context (GKE Autopilot, region, window, cadence) and summary metrics: accepted/exported throughput (means/peaks/totals), export efficiency (Eq. (1)), queue statistics, latency (p50/p95), drops, exporter errors, and resource footprint (CPU, memory, network).

IV. DISCUSSION

We evaluated a production-style OpenTelemetry (OTel) pipeline on GKE Autopilot under a sustained trace load and observed that exported throughput closely tracked accepted throughput, with deviations confined to a short saturation interval. We interpret this pattern as evidence that—in this configuration—scalability is governed primarily by exporter and queue policy rather than raw CPU or memory provisioning. We further examined the relationship between backpressure and tail latency: queue growth co-varied with p95 inflation and brief drops, and both subsided once pressure abated. This supports an operational focus on backpressure as the leading indicator of user-visible latency.

We operationalized these findings into concrete controls. First, we recommend alerting on the rate of change of the Collector queue ($\Delta Q/\Delta t$) and enforcing a rolling export-efficiency floor (as defined by Eq. (1)) alongside a p95 target; together, these detect incipient saturation while recovery is still possible via batching/backoff. Second, we found that endpoint/protocol misconfiguration (e.g., OTLP/HTTP vs. OTLP/gRPC) manifests as exporter errors and transient backlog; we therefore advocate a pre-flight verification step (receiver protocol, exporter health, synthetic canary). Finally, we distinguish windowed efficiency from explicit loss: accepted–exported differences over finite windows reflect both backlog dynamics and scrape alignment, whereas drop counters report irrecoverable loss. Reporting both perspectives yields actionable guidance on when to increase exporter concurrency, adjust batching, or shard collectors.

The findings in this study reflect a specific configuration: a single OTel Collector instance on GKE Autopilot, a homogeneous regional cluster, and a controlled, trace workload. While this environment models real production clusters, larger deployments with heterogeneous node pools, multi-region routing, or spike-driven workloads may exhibit different queue dynamics. Workloads with high cardinality, tail-heavy request patterns, or multi-collector routing may push exporter bottlenecks earlier. Therefore, while the results are indicative of OTel behavior under realistic conditions, they should be interpreted as lower-bound performance rather than universal guarantees. Extending this evaluation across diverse cluster sizes, workload mixes, and network conditions represent an important direction for future work.

V. CONCLUSION AND FUTURE WORK

We evaluated a production-style OpenTelemetry (OTel) pipeline on GKE Autopilot under sustained load, measuring end-to-end throughput, tail latency, and backpressure. Exported throughput closely tracked accepted throughput; a

brief saturated interval produced queue growth, transient drops, and p95 inflation that subsequently recovered. These findings indicate that, in this configuration, scalability is governed mainly by exporter/queue policy rather than raw CPU or memory, and that backpressure is a reliable leading indicator of tail behavior. The study meets the stated objectives: (i) demonstrates scalability in a real cloud setting (throughput and export efficiency), (ii) identifies integration sensitivities (protocol/endpoint) with observable signatures (errors, queue, drops), and (iii) provides computational profiling that links queue dynamics to p95 tails. Limitations include a controlled workload and a 30-minute window; heterogeneous, diurnal traffic may surface different behaviors.

Future work: evaluate multi-Collector sharding and exporter concurrency tuning; assess tail-based sampling and batching strategies; extend runs to multi-hour horizons and adverse network conditions; and introduce more realistic workload mixes to refine capacity guidance.

ACKNOWLEDGMENT

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Ministry of Science and ICT (Project No. RS-2024-00438551, 30%);

The National Research Foundation of Korea (NRF) grant funded by the Korean government (Project No. RS-2023-00208460, 30%); and the Korea Creative Content Agency (KOCCA) under the Copyright Technology Global Talent Development Program (Project No. RS-2025-02221620, 40%).

REFERENCES

[1] OpenTelemetry, "Overview," accessed Oct. 29, 2025. [Online]. Available: https://opentelemetry.io/docs/specs/otel/overview/

[2] Prometheus, "Overview," accessed Oct. 29, 2025. [Online]. Available: https://prometheus.io/docs/introduction/overview/

[3] Google Cloud, "GKE Autopilot overview," *Google Kubernetes Engine (GKE) Documentation*, accessed Oct. 29, 2025. [Online]. Available: *https://docs.cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview?*

[4] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013, doi: 10.1145/2408776.2408794.

[5] J. Thalheim *et al.*, "Sieve: Actionable Insights from Monitored Metrics in Microservices," in *Proc. Middleware 2017*, 2017. [Online]. Available: https://arxiv.org/abs/1709.06686

[6] L. Zhang *et al.*, "The Benefit of Hindsight: Tracing Edge-Cases in Distributed Systems," 2022. [Online]. Available: https://arxiv.org/abs/2202.05769

[7] O. B. K. Bhuian and K.-W. Park, "Deep dive into OpenTelemetry for evaluation of their observability in edge computing environment," in *Proc. 10th Int. Conf. Next Gener. Comput. (ICNGC 2024)*, 2024, pp. 161–164. Accessed: Oct. 29, 2025. [Online]. Available: http://syscore.sejong.ac.kr/~woongbak/publications/C92.pdf

[8] OpenTelemetry, "Specification status summary," accessed Oct. 29, 2025. [Online]. Available: https://opentelemetry.io/docs/specs/status/