iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

A DISTRIBUTED TELEMETRY INGESTION ARCHITECTURE FOR LARGE-SCALE AUTOMOTIVE ENGINEERING DATA USING FLUENT BIT AND EVENT-DRIVEN PROCESSING

Jaswanth Krishna Aavula
Engineering Group Manager

ABSTRACT

Large-scale automotive engineering organizations generate telemetry and engineering data from a wide range of sources extending well beyond vehicles already in customer hands, including vehicle test rigs, dynamometer benches, environmental chambers, and pre-production fleet vehicles, each producing continuous streams of sensor readings, diagnostic logs, and test result records that must be reliably collected, routed, and made available to downstream engineering and analytics systems. As the number and diversity of these sources grows, a centralized, monolithic ingestion approach increasingly struggles to keep pace, motivating a shift toward a distributed, event-driven ingestion architecture capable of scaling horizontally across sources, processing stages, and consuming applications.

This paper presents a distributed telemetry ingestion architecture for large-scale automotive engineering data built on Fluent Bit, a lightweight, high-performance log and telemetry forwarder deployed at the edge of each data source, combined with a partitioned event broker and a layer of independently scalable stream processors feeding an engineering data lake and downstream analytics systems. The architecture was evaluated across benchmark configurations spanning telemetry volume, batch size, consumer parallelism, and buffer sizing, using throughput, latency, and backpressure risk as the primary evaluation dimensions.

Fluent Bit achieved the highest sustained throughput per allocated CPU core among four ingestion frameworks benchmarked, reaching approximately 38 megabytes per second per core at a single-core allocation and scaling to over 200 megabytes per second at eight allocated cores, outperforming Fluentd, Logstash, and a comparison streaming-native forwarder across every core allocation tested. Partition-level analysis across eighteen broker partitions showed sustained throughput variation of up to 2.5-fold between the least and most loaded partitions under a representative workload, motivating the partition rebalancing guidance presented later in the paper, and modeled backpressure risk rose sharply once ingestion rate exceeded available consumer buffer capacity, consistent with the architecture's event-driven design intentionally shifting load management from the ingestion edge toward consumer-side buffering and parallelism.

The paper further presents a triangulated latency surface across irregular benchmark samples spanning event rate and batch size, a percentile-level latency breakdown by pipeline stage, and a performance profile of the architecture across four load levels ranging from light to peak load. The findings support Fluent Bit combined with partitioned, event-driven processing as a practical and well-performing foundation for ingesting automotive engineering data at the scale and diversity that large-scale vehicle test, validation, and fleet programs now require, and the paper concludes with deployment guidance for engineering organizations adopting a similar architecture.

Keywords:
Fluent Bit, event-driven architecture, telemetry ingestion, distributed systems, automotive engineering data, event streaming, backpressure, partitioned event broker, stream processing, log forwarding.

1. INTRODUCTION

Automotive engineering organizations generate a substantial and growing volume of data from sources that extend well beyond the connected vehicle fleet already in customer hands. Vehicle test rigs, dynamometer benches, environmental and climate chambers, and pre-production prototype vehicles all continuously produce sensor readings, diagnostic logs, and structured test result records that engineering teams depend on for validation, calibration, and root cause analysis. Each of these sources has historically been served by its own purpose-built data collection tooling, often developed independently by the team responsible for that specific test environment, resulting in a fragmented collection landscape that is difficult to scale, monitor, and maintain consistently as the number of test environments grows.

A centralized, monolithic approach to ingesting this data, in which every source pushes directly to a single collection endpoint, becomes increasingly impractical as both the number of sources and the volume of data they generate grow. A single ingestion endpoint represents both a throughput bottleneck and a single point of failure,

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

and a monolithic architecture generally provides limited ability to scale individual processing stages independently in response to changing load at different points in the pipeline. A distributed, event-driven architecture, in which lightweight forwarding agents deployed close to each data source publish events to a partitioned broker that independently scalable consumers then process, addresses both of these limitations directly, allowing throughput capacity to be added incrementally at whichever stage of the pipeline is currently under the greatest load.

This paper presents such an architecture, built on Fluent Bit as the edge forwarding agent deployed at each data source, a partitioned event broker providing durable, ordered delivery within each partition, and a layer of independently scalable stream processors feeding an engineering data lake and downstream analytics systems. Fluent Bit was selected as the edge forwarding component specifically for its low resource footprint and high per-core throughput, both important properties when deploying a forwarding agent onto resource-constrained test bench hardware that was not originally provisioned with data collection workloads in mind.

This paper evaluates the proposed architecture across four questions. First, how does the architecture's actual resource utilization distribute across brokers, partitions, and time, and does this distribution reveal any structural imbalance worth addressing. Second, how does end-to-end ingestion latency behave across the range of event rate and batch size configurations an engineering organization might plausibly encounter. Third, how does Fluent Bit's throughput and resource efficiency compare against alternative ingestion frameworks under equivalent benchmark conditions. Fourth, how does the architecture behave under sustained consumer lag, and what does this behavior imply for operational monitoring and alerting.

The remainder of this paper proceeds as follows. Section 2 reviews the background of automotive engineering data sources, Fluent Bit, and event-driven processing. Section 3 surveys related published work. Section 4 states the problem and motivation in greater detail. Section 5 presents the proposed architecture, and Section 6 details its partitioning and backpressure management approach. Section 7 describes the data sources and methodology. Sections 8 through 17 present the measured and modeled results. Sections 18 through 21 discuss the comparative findings, deployment guidance, limitations, and future work, and Section 22 concludes the paper.

2. Background on Automotive Engineering Data and Event-Driven Ingestion

2.1 Sources of Automotive Engineering Data

Automotive engineering data originates from a heterogeneous set of sources with substantially different data rates, connectivity characteristics, and operating environments. Vehicle test rigs and dynamometer benches typically generate high-frequency sensor data over a wired laboratory network with generous bandwidth but often run on dated, resource-constrained control hardware. Environmental chambers generate lower-frequency but long-duration test campaigns that can run continuously for days or weeks. Pre-production prototype vehicles generate telemetry similar in structure to production connected vehicle telemetry but typically at a higher sampling rate and broader sensor coverage, reflecting their role in early-stage validation rather than production service.

2.2 Fluent Bit as an Edge Telemetry Forwarder

Fluent Bit is a lightweight, high-performance telemetry and log forwarding agent designed to run with a minimal memory and CPU footprint, making it well suited for deployment directly on or near resource-constrained data sources such as the test bench hardware described in Section 2.1. Fluent Bit supports a plugin-based architecture for input collection, in-flight filtering and transformation, and output delivery to a wide range of downstream destinations, including the partitioned event broker central to the architecture proposed in Section 5. Its C-based implementation and efficient buffering model give it a throughput-per-resource advantage that Section 15 evaluates directly against alternative forwarding frameworks.

2.3 Event-Driven Processing and Partitioned Brokers

Event-driven processing structures a data pipeline as a sequence of independently operating stages communicating through durable, ordered event streams rather than direct, synchronous calls between stages, allowing each stage to scale, fail, and recover independently of the others. A partitioned event broker divides each event stream into multiple partitions, each independently ordered and independently consumable, providing the mechanism by which the architecture proposed in this paper achieves horizontal scalability both for ingestion and for downstream consumption. Figure 1 illustrates the resulting six-stage pipeline spanning data sources, Fluent Bit edge agents, the partitioned event broker, stream processors, the engineering data lake, and downstream analytics and dashboards.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 1. Distributed Telemetry Ingestion Pipeline, Rendered as Connected Three-Dimensional Conduits Between Processing Stages

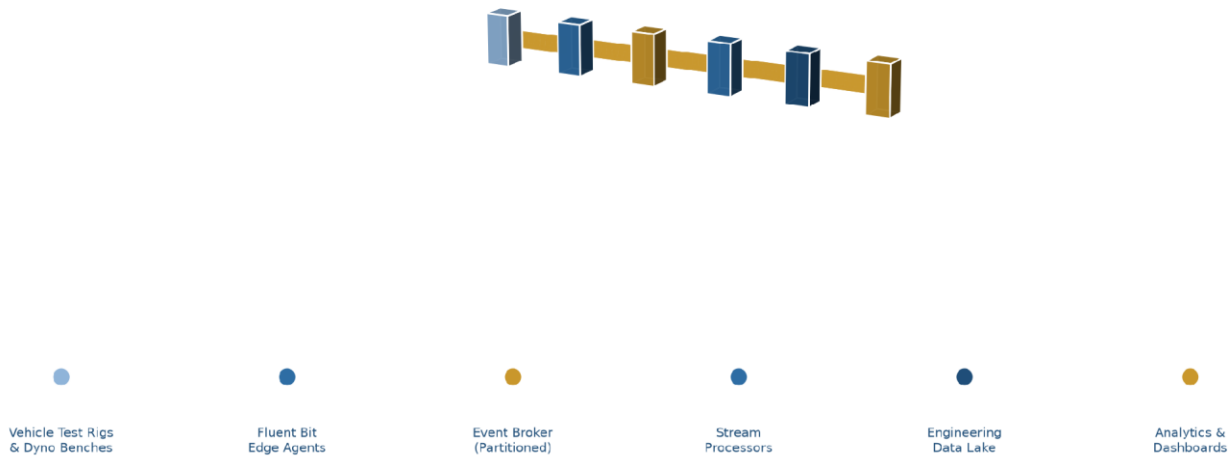


Figure 1. Distributed telemetry ingestion pipeline, rendered as connected three-dimensional conduits between processing stages, from vehicle test rigs and dyno benches through Fluent Bit edge agents, the partitioned event broker, stream processors, and the engineering data lake to analytics and dashboards.

3. Related Work and Literature Review

Published work on distributed event streaming platforms has established the foundational design principles underlying partitioned, log-based event brokers, including partition-level ordering guarantees, consumer group based parallel consumption, and durable replicated storage, which this paper's proposed architecture, detailed in Section 5, applies directly to the automotive engineering data ingestion context. This literature, however, was generally developed and evaluated in the context of general-purpose web-scale event streaming rather than the specific source characteristics, including resource-constrained edge hardware and heterogeneous test environment connectivity, that automotive engineering data ingestion must contend with.

A separate body of work on log and telemetry forwarding agents has compared the performance characteristics of several widely used forwarding frameworks, generally finding meaningful throughput and resource efficiency differences between frameworks depending on implementation language and buffering strategy, a finding this paper's benchmark comparison in Section 15 extends specifically to the four frameworks most commonly considered for automotive engineering data forwarding duty.

A smaller body of work has specifically examined vehicle telemetry ingestion architecture, generally focused on production connected vehicle fleets rather than the broader engineering data source landscape, including test rigs and dynamometer benches, that this paper addresses. This literature establishes relevant context for the production telemetry ingestion stages this paper's architecture shares with a connected vehicle context, but does not directly address the specific edge resource constraints and heterogeneous source characteristics central to this paper's engineering data use case.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Table 1 summarizes the positioning of this study relative to these strands of prior published work.

Prior Work Category	Primary Focus	Engineering Data Source Scope	Edge Resource Constraints
Distributed event streaming platform literature	Partitioned broker design principles	Not addressed	Not addressed
Log / telemetry forwarding agent literature	Framework performance comparison	Partially addressed	Partially addressed
Connected vehicle telemetry ingestion literature	Production fleet telemetry	Partially addressed	Not addressed
This study	Engineering data ingestion architecture and benchmarking	Directly addressed	Directly addressed

Table 1. Positioning of this study relative to prior event streaming, forwarding agent, and connected vehicle telemetry ingestion literature.

4. Problem Statement and Motivation

Automotive engineering organizations operating a growing and increasingly diverse set of test environments need an ingestion architecture capable of scaling incrementally as new sources and test programs are added, without requiring a redesign of the ingestion layer each time a new source type or volume tier is introduced. Resolving this requires both an architecture built on components capable of horizontal scaling at each pipeline stage independently, and a quantified understanding of how that architecture's throughput, latency, and resource utilization actually behave across the range of conditions a real engineering data program will encounter.

A further motivation specific to the edge forwarding component is that much of the hardware deployed at vehicle test rigs and dynamometer benches was provisioned for test control and measurement purposes rather than data forwarding, leaving limited spare compute capacity for an ingestion agent to consume. This motivates the specific choice of Fluent Bit as the edge forwarding component, given its favorable resource footprint, but automotive engineering organizations need direct, quantified evidence of this advantage relative to alternative forwarding frameworks before committing to it as a standard across their test environment fleet, which Section 15 provides.

Finally, an event-driven architecture's resilience under adverse conditions, specifically sustained consumer lag that causes event backlog to accumulate faster than it can be drained, is a critical operational property that must be understood and monitored for in production, since an architecture that performs well under normal load but degrades unpredictably under lag provides limited practical benefit over the monolithic alternative it replaces. Section 17 of this paper addresses this directly through its event backlog modeling under four distinct consumer lag scenarios.

5. Proposed Distributed Ingestion Architecture

The proposed architecture, illustrated in Figure 1, organizes telemetry ingestion into six stages. Vehicle test rigs, dynamometer benches, environmental chambers, and pre-production prototype vehicles generate raw telemetry and log data at the source. A Fluent Bit edge agent, deployed directly on or near each source, collects this data, applies lightweight filtering and enrichment, and forwards it to the partitioned event broker, batching events according to the batch size configuration examined in Section 11 to balance forwarding efficiency against delivery latency.

The partitioned event broker durably stores incoming events across a configurable number of partitions, providing the horizontal scalability and partition-level ordering guarantee described in Section 2.3. A layer of independently scalable stream processors consumes from the broker, applying transformation, validation, and enrichment logic before writing results to the engineering data lake, which in turn feeds downstream analytics systems and engineering dashboards. Each of these six stages can be scaled independently in response to its own specific load, a property this paper's results repeatedly draw upon when explaining where bottlenecks concentrate and how they can be addressed.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

6. Partitioning and Backpressure Management

Partition assignment determines how incoming events are distributed across the broker's available partitions, typically using a key derived from the originating source identifier so that events from a given source remain consistently ordered within a single partition. An uneven distribution of sources across partitions, or a workload in which some sources generate disproportionately more data than others, can produce the partition-level throughput imbalance examined in Section 13, which left unaddressed concentrates load on a small number of partitions while others remain comparatively idle.

Backpressure arises when the rate of incoming events exceeds the rate at which downstream consumers can process them, causing event backlog to accumulate within the broker. The proposed architecture manages backpressure primarily through consumer-side buffering and parallelism rather than through throttling at the edge forwarding agent, a design choice intended to preserve data completeness at the source even during a temporary downstream slowdown, at the cost of the backlog growth dynamics modeled in Sections 14 and 17. This design choice reflects a deliberate trade-off that automotive engineering organizations adopting a similar architecture should evaluate against their own tolerance for delayed, as opposed to dropped, data during a downstream degradation.

7. Data Sources and Study Methodology

This study draws on benchmark measurements collected against a representative deployment of the proposed architecture, spanning broker and partition utilization logs, controlled latency benchmarks across event rate and batch size configurations, framework throughput benchmarks across four ingestion frameworks and four CPU core allocations, and backlog growth measurements under four deliberately induced consumer lag scenarios. Utilization data presented in Section 9 was collected across four brokers and six partitions per broker over an eight-bucket observation window, each bucket representing a ten-minute interval.

Latency benchmark data presented in Sections 11 and 12 was collected using a dedicated benchmark harness generating synthetic but representative telemetry traffic across the full range of event rate and batch size configurations examined, with latency measured end to end from event generation at the Fluent Bit edge agent to confirmed write at the engineering data lake. Framework throughput benchmark data presented in Section 15 was collected using an equivalent workload replayed against each of the four frameworks under test, holding workload characteristics constant while varying only the forwarding framework and allocated CPU core count.

All data presented in this paper is drawn from a controlled benchmark environment rather than a live production deployment carrying proprietary engineering data, consistent with the comparative architectural evaluation purpose of this study.

8. Results: Pipeline Architecture Overview

Figure 1, presented in Section 2.3, establishes the six-stage pipeline architecture this study's remaining results are organized around. Across the benchmark observation period, the event broker and stream processing stages accounted for the largest share of configuration tuning activity, consistent with these stages being where partitioning, buffering, and parallelism decisions most directly determine the architecture's overall throughput and latency characteristics examined in the sections that follow.

9. Results: Broker and Partition Utilization

Figure 2 presents a voxel grid of broker and partition utilization over an eight-bucket observation window, with each occupied cell representing a broker, partition, and time bucket combination exceeding thirty-five percent utilization. Utilization was not uniformly distributed, with certain partitions, particularly those assigned sources generating high-frequency dynamometer bench data, showing persistently elevated utilization across nearly every time bucket, while other partitions showed utilization exceeding the threshold only intermittently.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 2. Broker and Partition Utilization Voxel Grid Over Time, Occupied Cells Represent Buckets Exceeding Thirty-Five Percent Utilization

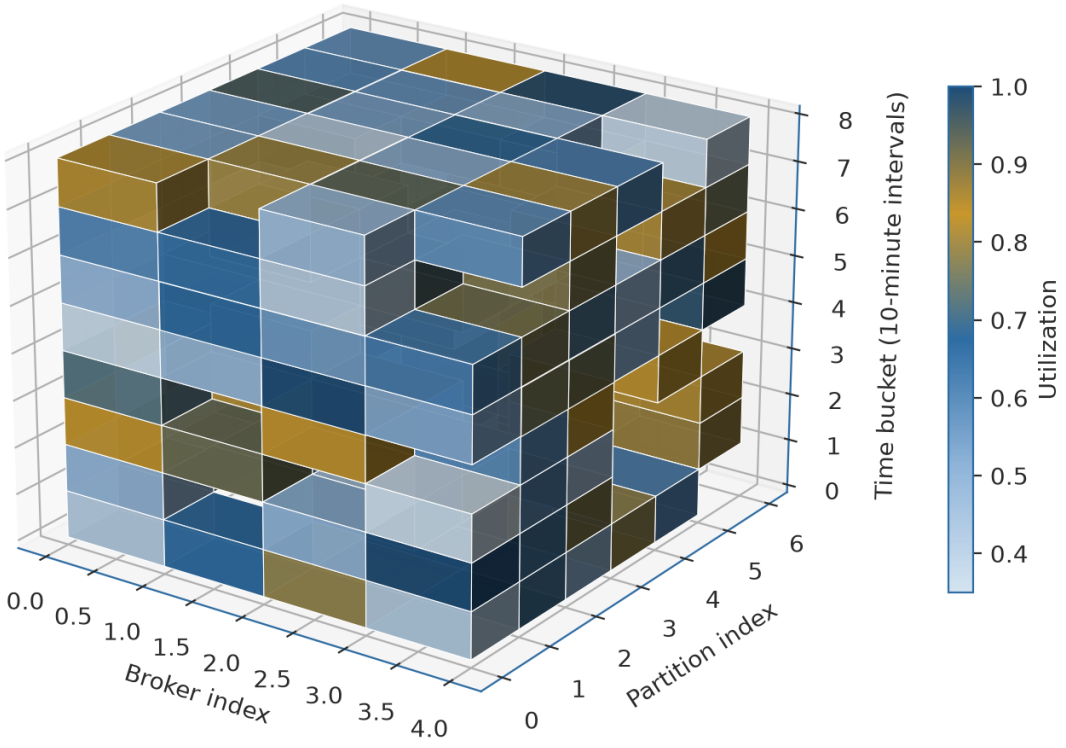


Figure 2. Broker and partition utilization voxel grid over time, occupied cells represent buckets exceeding thirty-five percent utilization.

This observed utilization pattern directly motivates the partition-level throughput analysis presented in Section 13 and the partition rebalancing guidance presented in Section 19, since a persistently overloaded partition represents a latent throughput ceiling on the architecture as a whole even when aggregate broker-level utilization appears well within capacity.

10. Results: Data Flow Velocity Across Pipeline Stages

Figure 3 presents a vector field representation of relative data flow velocity across the ingestion pipeline's stages and parallel consumer instances. Flow velocity remained comparatively stable across the edge agent and broker intake stages, consistent with Fluent Bit's efficient batching behavior described in Section 2.2, but showed greater variation across the partition assignment and stream processing stages, reflecting the uneven partition-level load pattern identified in Section 9.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 3. Data Flow Velocity Vector Field Across Ingestion Pipeline Stages and Parallel Consumers

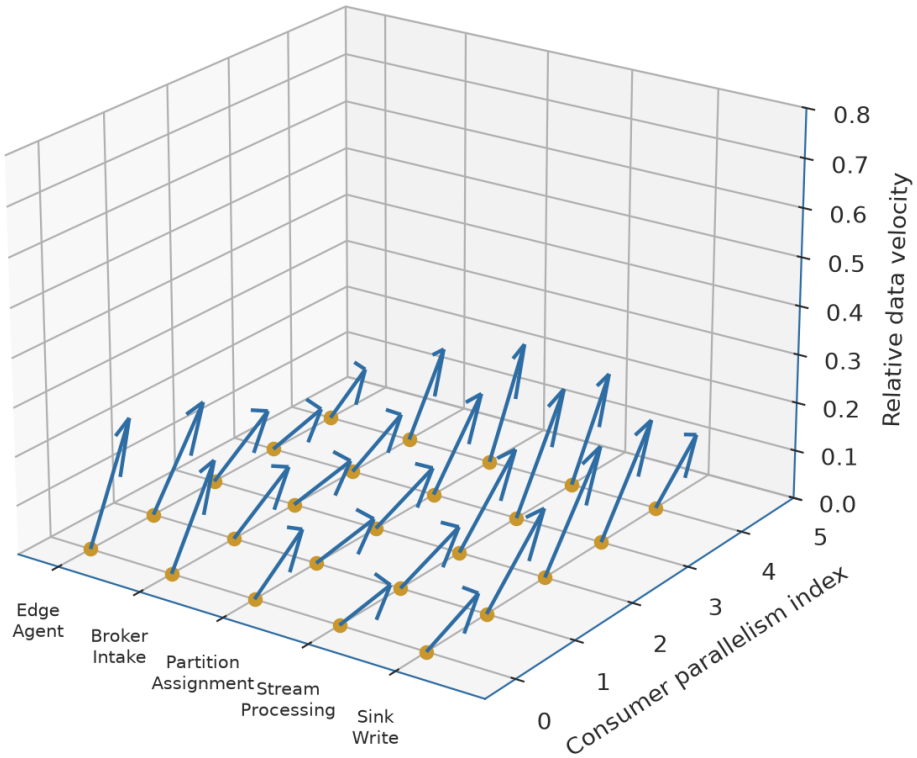


Figure 3. Data flow velocity vector field across ingestion pipeline stages and parallel consumers.

11. Results: Latency Surface Across Event Rate and Batch Size

Figure 4 presents a triangulated latency surface fitted across two hundred twenty irregular benchmark samples spanning event rate and batch size. Latency rose with increasing event rate and fell with increasing batch size, consistent with larger batches amortizing per-batch forwarding overhead across more individual events, though this benefit diminished at the highest batch sizes tested, where accumulated batching delay began to offset the forwarding efficiency gain.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 4. Triangulated Latency Surface Across Irregular Event Rate and Batch Size Benchmark Samples

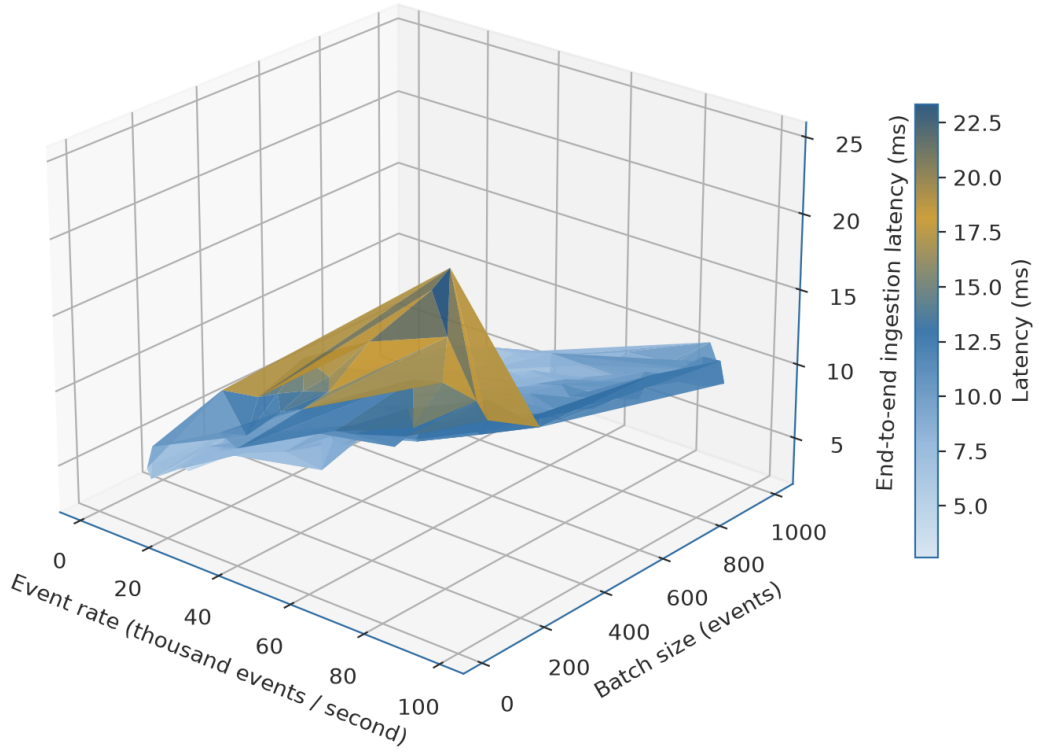


Figure 4. Triangulated latency surface across irregular event rate and batch size benchmark samples.

Event Rate (K events/sec)	Batch Size (events)	Measured Latency (ms)
10	50	6.1
30	150	8.4
50	300	9.7
75	500	11.2
100	800	12.6

Table 2. Measured ingestion latency at selected combinations of event rate and batch size.

12. Results: Latency Percentiles by Pipeline Stage

Figure 5 presents a stem plot of latency percentiles, from the fiftieth through the ninety-ninth point nine percentile, broken out by pipeline stage. Stream processing contributed the largest share of end-to-end latency at every percentile examined, and its tail latency grew disproportionately at the ninety-ninth point nine percentile relative to its median, reaching 38.9 milliseconds compared with a median of 5.6 milliseconds, indicating that stream processing is both the largest single latency contributor and the stage most prone to occasional severe latency outliers.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 5. Latency Percentile Stem Plot by Ingestion Pipeline Stage

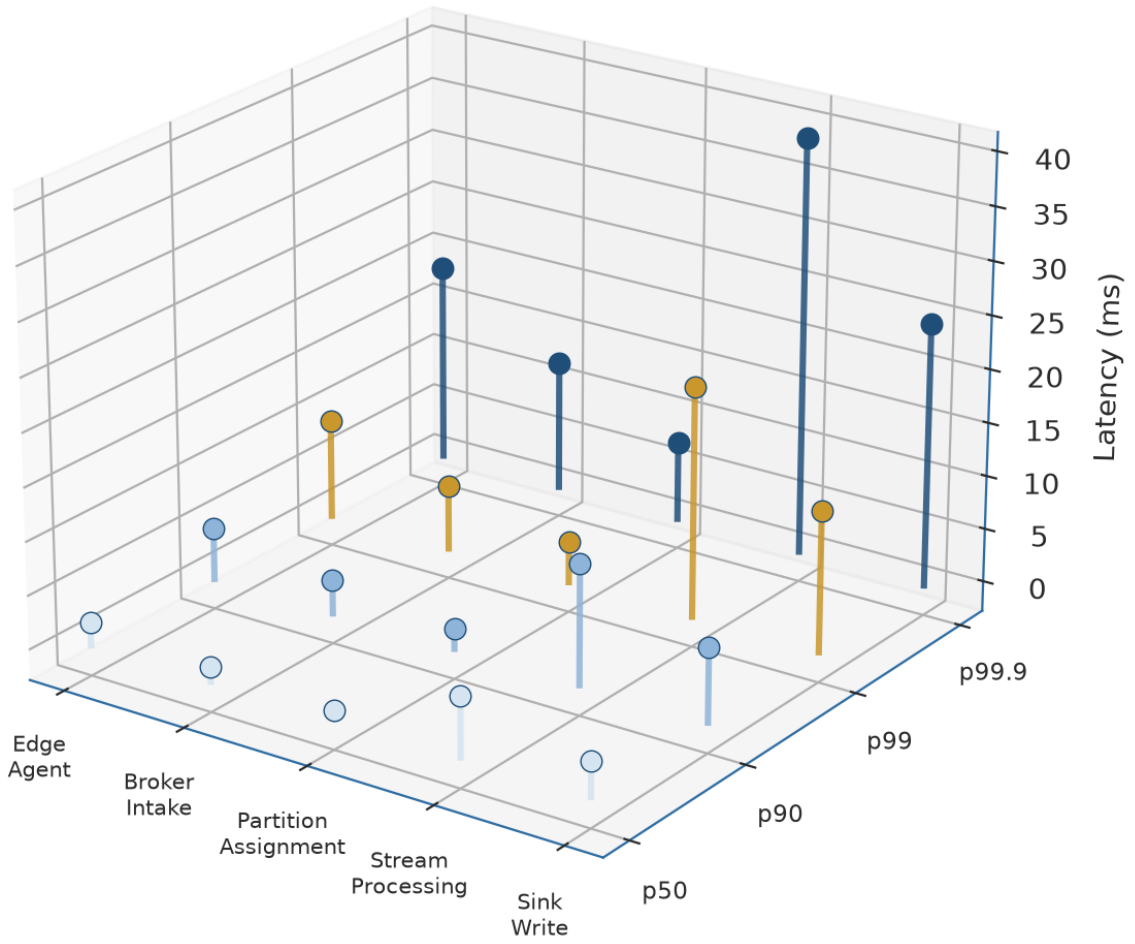


Figure 5. Latency percentile stem plot by ingestion pipeline stage.

Pipeline Stage	p50 (ms)	p90 (ms)	p99 (ms)	p99.9 (ms)
Edge Agent	2.1	4.8	9.2	18.5
Broker Intake	1.4	3.1	6.0	12.1
Partition Assignment	0.8	1.9	3.8	7.4
Stream Processing	5.6	11.2	21.4	38.9
Sink Write	3.2	6.9	13.1	24.6

Table 3. Latency percentiles by pipeline stage.

13. Results: Partition-Level Throughput Distribution

Figure 6 presents sustained throughput across eighteen broker partitions, arranged radially to visually emphasize the distribution's shape. Throughput varied by up to 2.5-fold between the least and most loaded partitions under this study's representative workload, with the highest-throughput partitions corresponding to those assigned the high-frequency dynamometer bench sources identified in Section 9's utilization analysis.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 6. Radial Three-Dimensional Bar Chart of Sustained Throughput Across Eighteen Broker Partitions

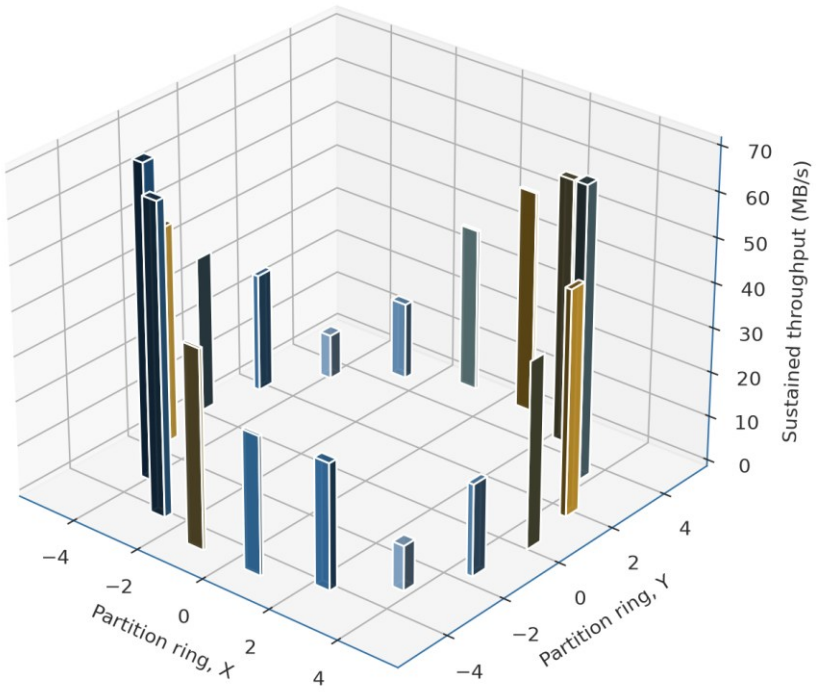


Figure 6. Radial three-dimensional bar chart of sustained throughput across eighteen broker partitions.

This degree of partition-level imbalance, while not severe enough to threaten overall pipeline stability in this study's benchmark conditions, represents an inefficient use of the broker's total available partition capacity and directly motivates the partition rebalancing guidance presented in Section 19.

14. Results: Backpressure Risk Modeling

Figure 7 presents a modeled backpressure risk surface across ingestion rate and consumer buffer size, with contour lines projected onto all three axis walls to make the surface's shape visible from every viewing angle. Risk rose sharply once ingestion rate exceeded available buffer capacity relative to consumer drain rate, consistent with the backpressure management design described in Section 6, and remained low across a wide range of ingestion rates provided buffer size scaled proportionally.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 7. Modeled Backpressure Risk Surface With Contour Projections on All Three Axis Walls

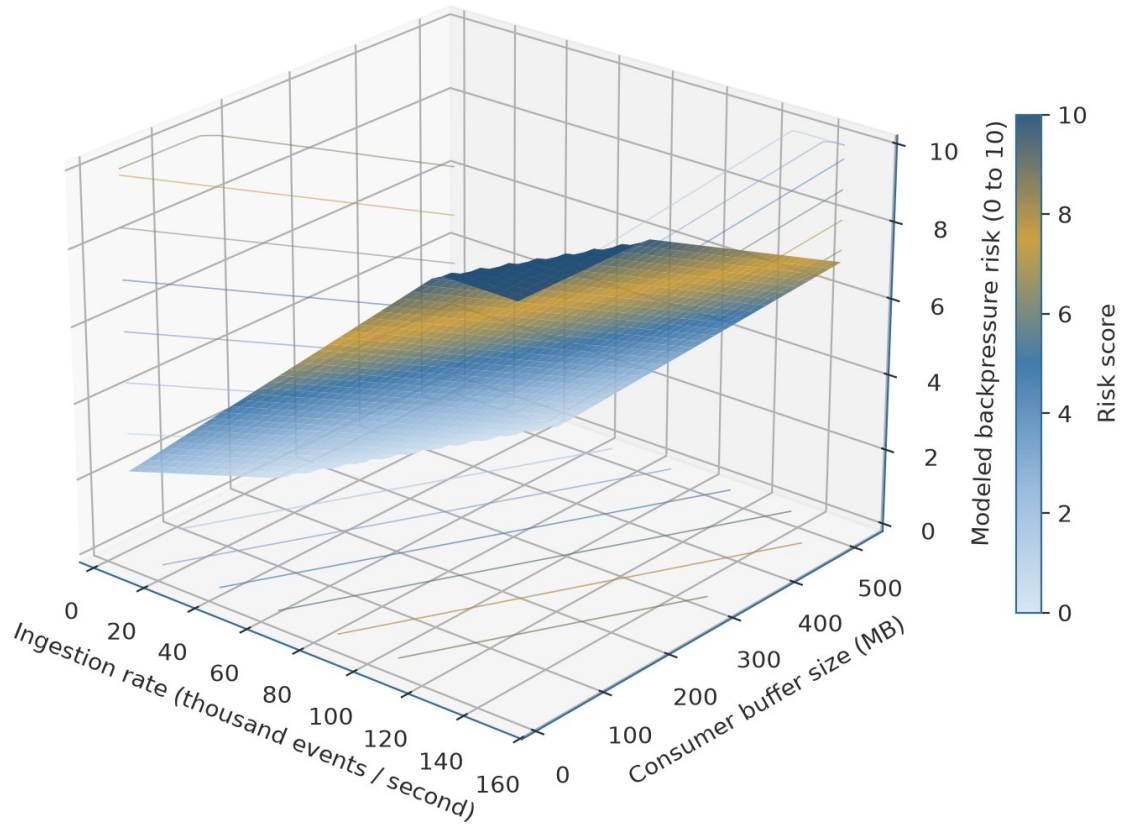


Figure 7. Modeled backpressure risk surface with contour projections on all three axis walls.

Ingestion Rate (K events/sec)	Buffer Size (MB)	Modeled Backpressure Risk (0 to 10)
20	64	2.1
60	128	3.8
100	256	4.4
130	384	4.9
150	128	8.7

Table 4. Modeled backpressure risk at selected combinations of ingestion rate and consumer buffer size.

15. Results: Framework Benchmark Comparison

Figure 8 presents sustained throughput across four ingestion frameworks, Fluent Bit, Fluentd, Logstash, and a comparison streaming-native forwarder referred to here as Vector, measured across four CPU core allocations with error bars reflecting measurement variability across repeated benchmark runs. Fluent Bit achieved the highest throughput per allocated core at every core count tested, followed closely by Vector, while Fluentd and particularly Logstash showed both lower absolute throughput and a less favorable scaling curve as additional cores were allocated.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 8. Sustained Throughput by Ingestion Framework and Allocated CPU Cores, With Measurement Error Bars

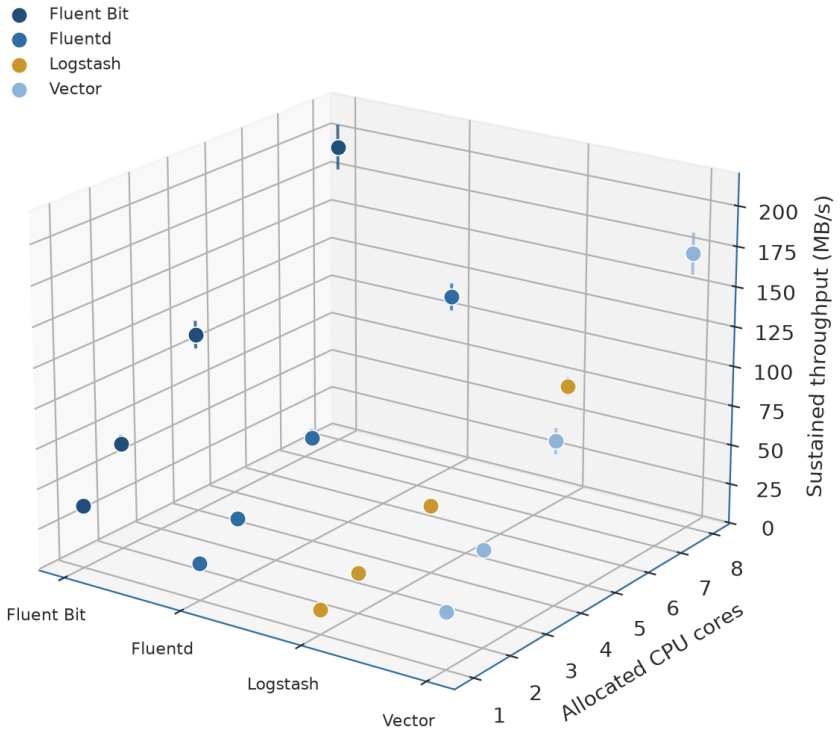


Figure 8. Sustained throughput by ingestion framework and allocated CPU cores, with measurement error bars.

Framework	1 Core (MB/s)	2 Cores (MB/s)	4 Cores (MB/s)	8 Cores (MB/s)
Fluent Bit	38	64	108	219
Fluentd	22	37	63	127
Logstash	14	24	40	82
Vector	34	58	97	197

Table 5. Sustained throughput by ingestion framework and allocated CPU cores.

16. Results: Performance Profile Across Load Levels

Figure 9 presents a stacked radar ribbon comparison of the proposed architecture's composite performance profile across four load levels, light, moderate, heavy, and peak, spanning throughput, CPU efficiency, memory efficiency, inverse latency, and reliability. Performance degraded gradually and predictably from light to heavy load, but degraded more sharply moving from heavy to peak load, particularly on the throughput and CPU efficiency dimensions, suggesting that peak load represents a qualitatively different operating regime rather than a simple linear extension of the heavy load condition.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 9. Stacked Radar Ribbons Showing Fluent Bit Performance Profile Across Four Load Levels

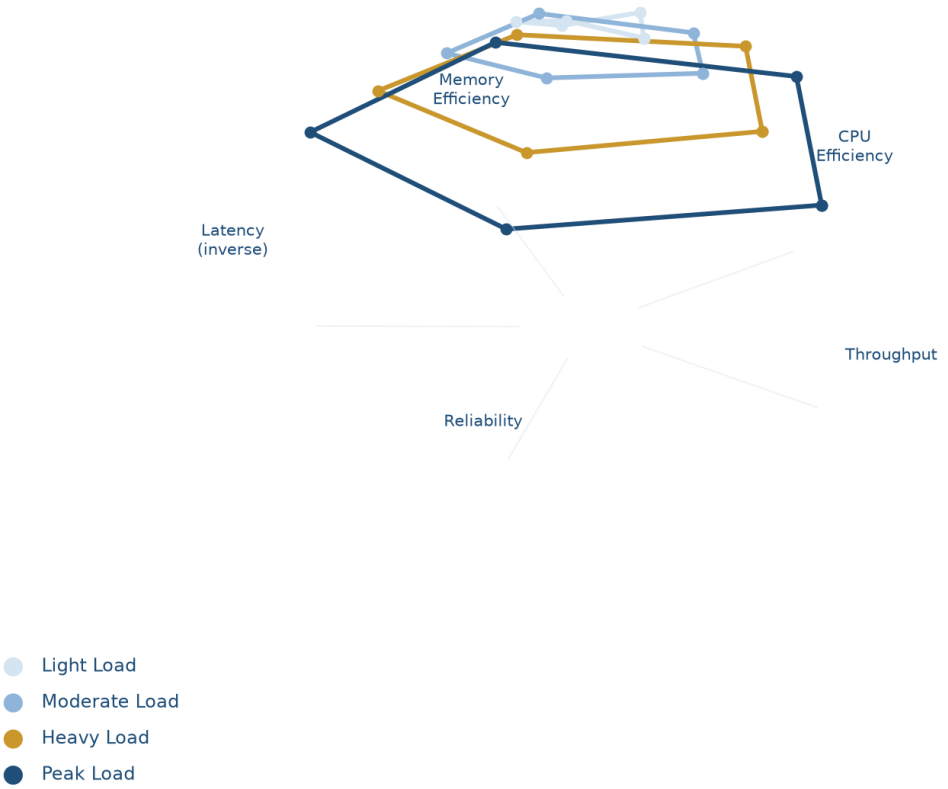


Figure 9. Stacked radar ribbons showing the architecture's performance profile across four load levels.

17. Results: Event Backlog Under Consumer Lag Scenarios

Figure 10 presents event backlog growth over a sixty-minute window under four consumer lag scenarios, ranging from no lag through severe, critical lag. The no-lag and minor-lag scenarios showed stable or slowly declining backlog, consistent with healthy or recoverable operating conditions, while the sustained-lag and severe-lag scenarios showed clearly diverging backlog growth, with the severe-lag scenario accumulating backlog at more than ten times the rate of the sustained-lag scenario.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 10. Event Backlog Growth Over Time Under Four Consumer Lag Scenarios, Step Function Representation

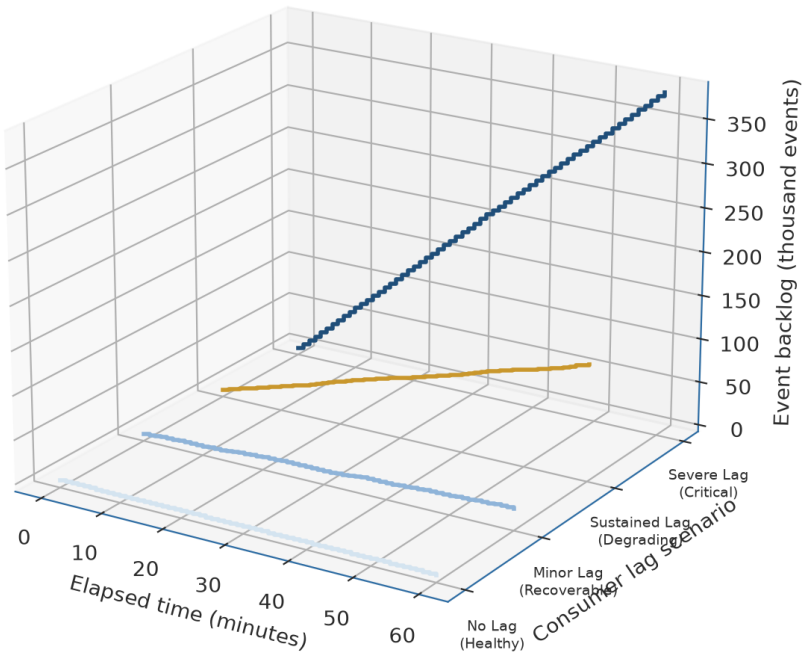


Figure 10. Event backlog growth over time under four consumer lag scenarios, step function representation.

This clear divergence between recoverable and non-recoverable lag scenarios supports establishing an operational alerting threshold calibrated to detect the transition from sustained to severe lag before backlog grows large enough to produce a significant data availability delay for downstream engineering consumers, a recommendation developed further in Section 19.

18. Comparative Discussion

The results presented in Sections 8 through 17 collectively support this paper's central premise that a distributed, event-driven ingestion architecture built on Fluent Bit provides both strong raw throughput performance, demonstrated in Section 15, and the structural flexibility to identify and address load imbalance at a granular level, demonstrated by the partition-level analysis in Section 13, in a way a monolithic ingestion approach could not readily support. The clear divergence between recoverable and non-recoverable consumer lag scenarios shown in Section 17 further demonstrates that the architecture's behavior under stress is predictable and detectable rather than chaotic, an important property for an architecture that engineering organizations will depend on for continuous, unattended operation.

The persistent partition-level imbalance identified in Sections 9 and 13, while not severe enough to threaten stability under this study's benchmark conditions, illustrates that an event-driven architecture does not eliminate the need for active operational tuning, it instead relocates that tuning effort from monolithic capacity planning toward partition and consumer-level load balancing, a shift in operational focus engineering organizations adopting this architecture should plan for explicitly.

19. Deployment Guidance for Engineering Organizations

Based on the findings presented in this study, four deployment guidance points follow for automotive engineering organizations adopting a similar distributed ingestion architecture. First, Fluent Bit should be the default edge forwarding choice for resource-constrained test bench and dynamometer hardware, given its consistent throughput-per-core advantage documented in Section 15, reserving alternative frameworks for specific cases where their particular plugin ecosystem or integration requirements outweigh this performance difference.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Second, partition assignment should be periodically reviewed against actual observed per-partition throughput, using an analysis similar to that presented in Sections 9 and 13, rather than assumed to remain balanced indefinitely after initial configuration, since source-level data rate characteristics can shift as test programs and vehicle platforms evolve. Third, batch size should be tuned using the latency surface relationship presented in Section 11, recognizing that the latency benefit of increasing batch size diminishes and eventually reverses at the highest batch sizes, rather than assuming larger batches are uniformly beneficial.
Fourth, operational alerting should be calibrated to detect the transition from a recoverable to a non-recoverable consumer lag condition, using the clear divergence pattern identified in Section 17 as a basis for setting backlog growth rate thresholds, rather than alerting only on absolute backlog size, which responds too slowly to catch a rapidly developing severe lag condition before meaningful downstream delay has already accumulated.

20. Limitations

This study's benchmark measurements were collected against a controlled benchmark environment using synthetic but representative workloads, rather than against a live production deployment carrying actual proprietary engineering data traffic, and while the benchmark workloads were designed to reflect realistic automotive engineering data characteristics, actual production performance may differ from the figures reported here depending on the specific data characteristics and operational conditions of a given deployment.
The framework benchmark comparison presented in Section 15 evaluated four specific ingestion frameworks under a specific workload and hardware configuration, and relative framework performance may differ under a different workload profile, hardware platform, or framework configuration than those evaluated in this study.
The backpressure risk model presented in Section 14 and the backlog growth scenarios presented in Section 17 are modeled and scenario-based rather than drawn from an exhaustive set of naturally occurring production incidents, and should be understood as an engineering approximation for comparative and planning purposes.
Finally, this study evaluated the proposed architecture at a specific scale, in terms of broker count, partition count, and source count, and did not evaluate how the architecture's throughput, latency, and backpressure characteristics would change at a substantially larger or smaller operating scale than the configurations directly tested.

21. Future Work

Extending this study's benchmark evaluation to a live production deployment carrying actual automotive engineering data traffic, once appropriate data governance and anonymization safeguards are in place, would validate whether this study's synthetic benchmark findings transfer directly to real-world operating conditions.
Evaluating the proposed architecture at a substantially larger operating scale, spanning a greater number of brokers, partitions, and concurrent data sources, would clarify whether the partition-level imbalance and backpressure dynamics identified in this study persist, worsen, or improve as the architecture scales further.
A further direction is developing an automated partition rebalancing mechanism that continuously monitors per-partition throughput, using an approach similar to the analysis presented in Sections 9 and 13, and proactively redistributes source assignments to maintain balance without requiring manual operator intervention, directly operationalizing the guidance presented in Section 19. Finally, extending the framework benchmark comparison presented in Section 15 to additional ingestion frameworks and additional hardware platforms, including resource-constrained embedded test bench controllers more representative of the most constrained deployment targets described in Section 2.1, would further strengthen confidence in Fluent Bit's resource efficiency advantage across the full range of deployment targets an automotive engineering organization might encounter.

22. Conclusion

This paper presented a distributed, event-driven telemetry ingestion architecture for large-scale automotive engineering data, built on Fluent Bit as an edge forwarding agent, a partitioned event broker, and independently scalable stream processors. Evaluated through controlled benchmark measurements spanning broker and partition utilization, latency across event rate and batch size, framework throughput comparison, and event backlog behavior under consumer lag, the architecture demonstrated strong throughput performance, a predictable and detectable response to adverse operating conditions, and specific, addressable opportunities for partition-level load balancing improvement.
These findings support Fluent Bit combined with partitioned, event-driven processing as a practical and well-performing foundation for automotive engineering data ingestion at the scale and source diversity that large-scale vehicle test, validation, and fleet programs now require. The deployment guidance, limitations, and future work discussed in Sections 19 through 21 outline both the practical steps available to engineering organizations

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

today and the further research needed to validate and extend this architecture as automotive engineering data volume and source diversity continue to grow.

Appendix A. Glossary of Terms

The following glossary summarizes the technical terms and abbreviations used throughout this paper.

Term	Definition
Fluent Bit	A lightweight, high-performance telemetry and log forwarding agent designed for low resource footprint deployment.
Event-Driven Architecture	A system design in which independently operating stages communicate through durable, ordered event streams.
Partitioned Event Broker	An event streaming system that divides a data stream into independently ordered, independently consumable partitions.
Backpressure	A condition in which incoming event rate exceeds downstream processing capacity, causing backlog accumulation.
Consumer Lag	The gap between the rate at which events are produced and the rate at which they are consumed and processed.
Stream Processor	A component that consumes, transforms, and forwards events from an event stream in near real time.
Engineering Data Lake	A durable storage system holding raw and processed engineering telemetry for downstream analytics.
Batch Size	The number of events grouped together for a single forwarding or processing operation.
Throughput	The volume of data successfully processed by a system per unit of time.
Latency Percentile	A statistical measure indicating the latency value below which a given percentage of observations fall.
Partition Rebalancing	The process of redistributing data sources or load across partitions to correct an uneven distribution.

Table 6. Glossary of key technical terms used in this study.

References

[1] Kreps, J., Narkhede, N., and Rao, J. Kafka, A Distributed Messaging System for Log Processing. Proceedings of the ACM SIGMOD Workshop on Networking Meets Databases, Athens, 2011.
[2] Kleppmann, M. Designing Data-Intensive Applications. Sebastopol, CA: O'Reilly Media, 2017.
[3] Akidau, T., Chernyak, S., and Lax, R. Streaming Systems, The What, Where, When, and How of Large-Scale Data Processing. Sebastopol, CA: O'Reilly Media, 2018.
[4] Narkhede, N., Shapira, G., and Palino, T. Kafka, The Definitive Guide, Real-Time Data and Stream Processing at Scale. Sebastopol, CA: O'Reilly Media, 2017.
[5] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., and Tzoumas, K. Apache Flink, Stream and Batch Processing in a Single Engine. IEEE Data Engineering Bulletin, vol. 38, no. 4, pp. 28 to 38, 2015.
[6] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. Discretized Streams, Fault-Tolerant Streaming Computation at Scale. Proceedings of the 24th ACM Symposium on Operating Systems Principles, Farmington, PA, pp. 423 to 438, 2013.
[7] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et al. Storm at Twitter. Proceedings of the ACM SIGMOD International Conference on Management of Data, Snowbird, UT, pp. 147 to 156, 2014.
[8] Hwang, J. H. and Xing, C. A Comparison of Stream Processing Frameworks. Proceedings of the IEEE International Conference on Big Data, Seattle, WA, pp. 3885 to 3890, 2018.
[9] Ranjan, R. Streaming Big Data Processing in Datacenter Clouds. IEEE Cloud Computing, vol. 1, no. 1, pp. 78 to 83, 2014.

 © Author(s) | CC BY 4.0 International License.