International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 1465-1471
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Hybrid Sharding and Real-Time Thresholding: Scaling Tail-Latency Diagnostics for Global Traffic

Varun Raj*

Independent Researcher, USA
* **Corresponding Author Email:** varunrajswe@gmail.com - **ORCID:** 0000-0002-5007-8850

Article Info:

DOI: 10.22399/ijcesen.5053
Received : 11 January 2026
Revised : 01 March 2026
Accepted : 08 March 2026

Keywords

Tail Latency Optimization,
Distributed Systems Monitoring,
Hybrid Sharding Architecture,
Real-Time Performance Thresholding,
Conditional Profiling Mechanisms

Abstract:

High-scale distributed systems face significant challenges in understanding performance degradation at the tail end of latency distributions. In traditional performance monitoring methods, it is common for the average of all requests to mask the true experience users have when making the requests towards the outer edges of the distribution. Monitoring methods will generally either fail to identify rare instances of performance anomaly or consume too much computational resource when profiling regular transactions. This article introduces an advanced telemetry architecture, which provides detailed insights into tail latency without decreasing system efficiency. The proposed architecture utilizes a centralized controller to maintain continuously updated dynamic percentile cutoffs for thousands of concurrent experiments, which operate across rolling time windows. The use of minimal reporting means that the amount of information exchanged between the serving tasks and the controller will be very small in terms of volume. In addition, the controller's conditional profiling activates the detailed diagnostics collection mechanism only when latency threshold limits are exceeded. The architecture has been designed to accommodate hybrid sharding schemes that define different traffic profiles in global deployments. Multi-slice sharding allows for horizontal scaling and increased metric storage, while maintaining centralized co-ordination to accurately manage thresholds. The real-time monitoring of performance in both production environments and load-test environments improves the speed to deliver developer velocity. The presented work demonstrates that advanced observability approaches are capable of working in an efficient manner at a planetary scale if they are developed using outcome-aware principles, along with effective resource allocation that leverages the system resources intelligently.

1. Introduction

Average latency metrics often hide the real user experience in high-scale distributed systems. The percentile represents the most critical performance optimization area for organizations serving billions of requests. This is the "long tail" of latency distribution.

Identifying why a high percentile request is slow requires deep diagnostic data. Such data collection has high overhead. Traditional uniform sampling fails here. It either misses rare percentile events or wastes resources profiling healthy requests.

Tail-tolerance has emerged as a fundamental systems principle rather than merely a metric to optimize [1]. The focus shifts from minimizing average latency to ensuring consistent response times across all requests. This paradigm recognizes that user experience degrades significantly when even a small fraction of requests experience delays.

Modern datacenter networks demonstrate the trade-offs between bandwidth and latency. Through the use of Ultra-Low Latency versus minimal Bandwidth, the ability to use trading strategies within a distributed environment has significant advantages [2]. The design of network fabrics with latency as a priority increases the response capability of distributed systems. The same applies to the use of monitoring technology that selectively collects data to reduce the overall load on the network.

This article presents a sophisticated telemetry system designed for surgical visibility into tail latency. TThe architecture of the system guarantees that performance monitoring will be outcome-aware on a global scale, while still being able to perform the same functions as traditional diagnostic systems.

2. Motivation: The Precision Gap in Tail-Latency Monitoring

Standard requests and percentile requests often follow entirely different code paths. They encounter different resource bottlenecks. This creates a fundamental challenge in performance engineering.

Simple random sampling proves insufficient for tail latency understanding. Profiling small fractions of all traffic rarely captures enough tail-end data for statistical significance. Currently, it is very difficult to acquire the data needed by the Data Acquisitions Systems for troubleshooting.

The complexity of performance monitoring is exposed by interactivity measurements captured from web pages [3]. Time-to-interactivity metrics require tracking multiple resource loading stages. Each stage contributes differently to perceived responsiveness. Traditional averaging obscures the dependencies that create tail latency.

Geo-distributed machine learning systems face similar challenges at planetary scale [4]. Network latency between datacenters dominates training performance. Synchronization barriers amplify tail latencies across distributed workers. A single slow worker delays the entire training iteration.

A system must be outcome-aware to understand tail latency properly. It needs to decide whether to collect detailed diagnostics based on actual request latency after completion. This requires a real-time feedback loop.

The system can accommodate thousands of concurrent experiments, with each having a different performance profile, while traditional monitoring systems do not support these requirements. Static sampling rates are not capable of adapting to dynamic traffic patterns.

3. Solution: Intelligent Thresholding Using a Centralized Control System

Using a centralized controller to manage dynamic percentiles. Cut-offs for all active experiments are maintained by the centralized controller. It operates over a rolling time window.

3.1 Minimalist Reporting Architecture

The serving task sends a lightweight remote procedure call at the end of every request. This call contains only the experiment identifier and observed latency. This design minimizes memory and network load.

Efficient packet scheduling principles guide the reporting mechanism [5]. Software-based packet scheduling can enable microsecond-level latency with minimal CPU use, due to the use of priority queuing. In a priority queuing environment, Control Plane messages are given a higher priority than bulk Data Transfer. The system uses a weighted fair queuing mechanism to provide a configurable allocation of bandwidth. Diagnostic Reporting Traffic is guaranteed a fixed amount of bandwidth to ensure that monitoring information reaches the controller, even during periods of high traffic. Use of Queues for Control Message Management greatly reduces the time Control Messages are buffered before being sent to the Control Plane. The controller compares each request's latency against its current percentile cutoff. This comparison happens for each specific experiment. The system makes a binary decision based on this comparison using optimized hash table lookups.

3.2 Conditional Profiling Mechanism

The system triggers detailed metric transmission only when latency crosses the percentile threshold. Microstack traces and sub-task latency breakdowns are included in the metrics. Selective collection of Cache Miss Data. The Profiling Strategy utilizing Remote Direct Memory Access (RDMA) Congestion Control [6] feeds into the Profiling Strategy. The performance characteristics of large RDMA deployments are relevant to diagnostics [7]. The Congestion Notification Packet generation operates at fixed intervals of 50 microseconds to balance feedback responsiveness with processing overhead. When using the Reaction Point algorithm, the minimum timer value assigned for all updates is 55 microseconds. Rate control is implemented through the algorithm via exponential increment and multiplicative decrement. If a sending device detects congestion as a result of packets being marked, the sending rate will be reduced based on an alpha reduction factor. The current sending rate will be updated to reflect this change using the equation $RC = RC(1 - \alpha/2)$, with α starting at one. Thus, the Algorithm provides a more aggressive initial backoff that will taper off with time.

The Fast Recovery state continues for five consecutive iterations prior to switching to an Additive Increment. Each iteration increases in rate by 40 Mbps (or when data is transferred, the sending rate increases). The Byte Counter will trigger the increase in sending rate when 10 MB of data has been transmitted. With the 55 microsecond timer as an additional constraint, these factors facilitate rapid convergence toward reaching an equalisation of available bandwidth. The conditional approach will result in a major reduction in the amount of diagnostic overhead required. Only those requests that are out of the ordinary experience the same level of profiling as was incurred on them originally. Routine requests will proceed through the process with minimal instrumentation. The RDMA principles were applied to facilitate prioritisation of diagnostics data transmission. The feedback loop operates continuously with hardware support for marking congested flows.

3.3 Data Persistence and Query Layer

High-fidelity metrics are dumped periodically into an online analytical processing database. This enables complex historical trend visualization. Support For Multi-dimensional Querying The database has sophisticated multi-dimensional querying capabilities. Two important components are switch buffer thresholds. Switch buffer thresholds are essential to guarantee proper operation. The Headroom Buffer is set at 22.4 kilobytes (KB) per port on a per priority basis for packet buffering of "in-flight" packets. The ECN priority flow control threshold is the maximum of 24.47KB. The dynamic thresholds of the ECN marking mechanism utilize a Beta factor of 8.0 in the ECN threshold calculation and maintain the ECN marking threshold

below 21.75KB. The ECN marking mechanism maintains a minimum of 5KB and maximum of 200KB with a maximum marking probability of 1 percent creating a gradual slope for marking, which improves fairness among competing flows. The Alpha parameter's update uses a value of 1/256 for the stability "g" value. Batch write operations allow for the balancing of write amplification and maximization of both data freshness. Compression dramatically reduces storage size requirements compared to storing raw metric data. The system retains all the details of the generated metrics and has configurable retention policies in place to meet the operational needs of the customer. An in-memory persistence layer provides lightweight query interfaces. Developers can issue queries for specific experiment ID's instantly from the interface and receive performance distributions in real-time. Memory-mapped file access allows for low latency queries of recent metric data.

Index structures enable rapid filtering across multiple dimensions. Bloom Filters help eliminate extra disk accesses when searching for an identifier that doesn't exist. Range queries based on timestamps take advantage of being sorted so a quicker scan can be done than if they weren't sorted. A single controller instance can execute several queries simultaneously without affecting performance.

4. Evolution: Multi-Slice Sharding for Global Traffic

A single monolithic persistence layer became a bottleneck as traffic volume increased. Traffic diversity also grew. The system needed to differentiate between latency profiles for specific traffic slices.

Mobile web and desktop web traffic have different performance baselines. These differences require separate tracking. A uniform approach proves inadequate.

4.1 Hybrid Sharding Strategy

The architecture evolved to shard the in-memory persistence layer. Sharding is based on experiment identifiers. This allows horizontal scaling of data-heavy metrics storage.

Flash Caching Strategies will be reflected in Sharding Design. Kangaroo demonstrates that it is possible to store billions (not exactly) of small objects (not exactly) in 7.0 Bits of DRAM per object (not exactly).

Using a partitioned structure allows for a decrease in miss ratios of 29% compared to using traditional caching techniques. The index is broken into 64 separate partitions. Each of these partitions works as its own cache and contains both a flash log and a DRAM index.

The index further splits into 1,048,576 separate tables for granular sharding. This hierarchical partitioning reduces the flash offset size by 6 bits per object. The tag size decreases from 29 bits to 9 bits through table-based sharding. Next-pointer requirements drop from 64 bits to 16 bits by using offsets within table memory.

Set-associative cache structures enable rapid lookup operations with 4 KB sets. Each set holds approximately 40 objects for 100-byte object sizes. Using Incremental Segment Flush, the System can Work At 80% To 95% Capacity. Flash capacity utilization reaches 93% of total device space.

The controller remained unsharded for specific logic blocks. This maintains a consistent global view of experiment thresholds. The combination of local efficiency and global coordination is what enables the hybrid design to function seamlessly. Flash persists the data longer than would be possible if it were stored on memory.

Bloom Filters perform membership tests that are very efficient (approximately 10% false positive). These filters consume 3 bits of DRAM per object. The combined DRAM overhead for indexing and filtering totals 7.0 bits per object, representing a 4.3× improvement over prior state-of-the-art designs [7].

4.2 Sliced Metric Computation

The sharded architecture enables high-speed metric computation across diverse global traffic. Each shard processes its own allocation of traffic independently; therefore, the system will have a combined throughput of up to 158,000 requests per second. The controller collects information from all of the shards, aggregates it, and presents a single source of data.

The write amplification factor decreases dramatically through the sharded design. Kangaroo achieves application-level write amplification of approximately 5.8× compared to 17.9× for pure set-associative designs. This represents a 3.08× reduction in bytes written to flash. The log-structured component writes data sequentially in large segments, approaching 1× write amplification for logged data.

High Performance I/O Systems Will Reduce Latency In Data Processing [8]. The direct memory access path from the storage to the network means that CPU intervention can be minimized. The collection of metrics will be done Asynchronously so will not block for any other operation. The elimination of unnecessary data transfers is accomplished through the use of zero-copy data paths.

The write speed of the device is limited to 62.5 MB/s by using a log-structured cache layer with a capacity of around 5% (0.095GB) of the available storage for writing data. The cache layer will use the remaining 95% of the total flash capacity (1.199TB) to store the other data being written to the flash device and therefore allows better balancing of efficiency with total storage capacity.

By implementing an admission policy based on a threshold, which allows the system to only admit objects after sufficient hash collision has occurred, the number of writes to a flash device can be greatly decreased. By using a threshold of 2 object per set, the amount of flash writes will decrease by 32% while only slightly increasing the number of misses by 6.9%.Objects that receive hits during their time in the log are readmitted rather than discarded, maintaining effectiveness for popular items.

This design avoids centralized coordination bottlenecks. It maintains a high level of precision for calculating thresholds. The number of partitions for an object can be increased as needed to support increased volume for an

object. This provides a linear scaling relationship between the growth of the volume of traffic and the ability to support that traffic.

5. System Performance and Evaluation

The application supports multiple data center locations globally. The capacity of each data center is capable of processing a large number of requests per second while maintaining low overhead to do so.

Receiver-driven transport protocols (RDTPs) have been shown to have the ability to achieve very low latencies [9]. Homa achieves median message completion times of 4.7 microseconds for 100-byte remote procedure calls on unloaded 10 Gbps networks. The protocol maintains 99th percentile round-trip times below 15 microseconds at 80% network load. Using priority-based scheduling of 8 different priority levels to send short messages ahead of longer message transfers has reduced the latency of transport processes in comparison to conventional transport methods by orders of magnitude. Short messages are able to preempt the longer message transfers, thus reducing queuing delays. The system also employs an efficient method for managing buffers that will prevent congestion, while keeping the average queue lengths between 1 to 17 kB at 80% of the maximum load of the available network across the multiple switch egress ports.

Maximum observed queue length reaches 146 kilobytes in top-of-rack to host downlinks. Overcommitment mechanisms account for up to 56 kilobytes of buffer usage across scheduled priority levels. The system successfully handles several thousand simultaneous operations without performance degradation when incast control is enabled. Without incast control, the system maintains stability for approximately 300 concurrent operations before experiencing packet drops.

In-kernel storage functionality provides for the efficient storage of metrics; the execution of Extended Berkeley Packet Filter (eBPF) applications allows for the testing of diagnostics without the need for context switching.

The complete storage stack overhead totals 3.046 microseconds, comprising 48.6% of the 6.27 microsecond total latency for 512-byte random reads. The breakdown of overhead is as follows: 2.006 µs of file system overhead, 379 ns of block layer overhead, 351 ns kernel crossing overhead, 199 ns of system call overhead, and 113 ns of NVMe driver overhead. Of the total latency, the storage device contributes 3.224 µs (51.4%) of the overall latency.

The diagnostic system achieves substantial performance improvements through selective profiling. The amount of Increase in Throughput for Reads over Typical Read Operations was 47%-to-94%. Latency reductions of 20% to 34% are observed at the 99th percentile. Single-threaded lookup operations demonstrate up to 2.5× speedup compared to normal system calls. With storage structures requiring six levels of index traversal, average lookup latency decreases from 48.8 microseconds using read calls to 29.3 microseconds with optimized resubmission.

The metadata digest lookup mechanism operates with 96 nanoseconds average latency. The lockless design uses read-copy-update for concurrent access without blocking. BPF function verification completes within seconds during initial load. The scratch buffer allocates 4 kilobytes per request for intermediate data storage. A single resubmission permits a maximum of 16 physical segments to be simultaneously accessed.

Storage device capabilities promote high-bandwidth activity. Intel Optane devices deliver up to 7 gigabytes per second bandwidth with latencies as low as 3 microseconds. The combination of fast storage hardware and optimized software stack makes submicrosecond diagnostic overhead achievable.

The online analytical processing database ingests massive volumes of diagnostic data daily. Query response times remain swift for typical performance investigations. A developer can drill down into a particular experiment in a matter of seconds. Indexing access patterns allows for filtering of experiments by experiment identifier without delay.

When developers load tests utilize a load test integration, they receive rapid feedback regarding their performance regressions. Those engineers see their percentile profiles reflect the same as their production telemetry. This prevents performance issues from reaching production environments. Synthetic traffic generation replicates production characteristics with configurable arrival rates following Poisson distributions.

The system currently monitors thousands of concurrent experiments. Every experiment maintains separate percentile thresholds. The controller continuously updates those percentile thresholds based on rolling windows that span several minutes so that throughout a particular day, the adaptive threshold calculation takes into account diurnal variations in traffic as well as workload differences due to different slices of traffic.

6. Implementation Considerations

To maintain high availability, the centralized controller is a critical part of the system. The implementation of redundancy mechanisms allows for a centralized controller to function continuously, even in the event of a datacenter outage. Datastore replication occurs across multiple geographical locations to maintain data availability.

To communicate between nodes in a datacenter, the remote procedure call (RPC) protocol is used, which utilizes a binary serialized format. This allows for a low-bandwidth solution for the lightweight monitoring and reporting capability. Protocol buffers allow for backward compatibility with the schema as it evolves. Schema Versioning allows for gradual implementation of Monitoring Enhancements.

It is necessary to Manage the Storage or Persistence layer's Memory of the system. In order to achieve optimum memory optimization, the system uses bounded memory usage for an arbitrary traffic workload. To facilitate this, circular buffers have been implemented to provide a constant time insertion and deletion mechanism for expired data; lock-free data structures are used to minimize contention in multi-threaded scenarios.

The selection of a sharding key is vital to achieving balanced load distribution amongst the nodes in the

datacenter. Experiment identifier types provide an almost natural mechanism for partitioning by providing predictable cardinality. A Hash-Based Method is the means to distribute the Load Evenly over Shards. The hashing methodology provides a way to add new shards dynamically without requiring all of the data to be moved from one shard to another.

7. Operational Insights

The design of interfaces to integrate with the current monitoring infrastructure was particularly important. This system provides standard metrics endpoints that can be used by existing dashboards. The same thresholds that were used for conditional profiling of alerts will be available to be used as the basis for alert configuration.

The debugging capabilities of the system extend beyond just performance monitoring; the detailed metrics that are collected will allow for additional root cause analysis of the intermittent failures that may occur due to concurrency issues or through lock contention patterns that have been identified via the aggregation of other measurement data.

The system has already assisted with future infrastructure capacity planning because of this. Daily percentile historical data is useful for making infrastructure scaling decision. The difference between traffic slices can assist with growth trend analysis among different users and statistical models can use past data to allocate capacity proactively.

Developer adoption accelerated due to the load-test integration. Engineers trust the system because test environment data matches production format. This eliminates surprises during production rollouts. Continuous integration pipelines incorporate latency regression testing.

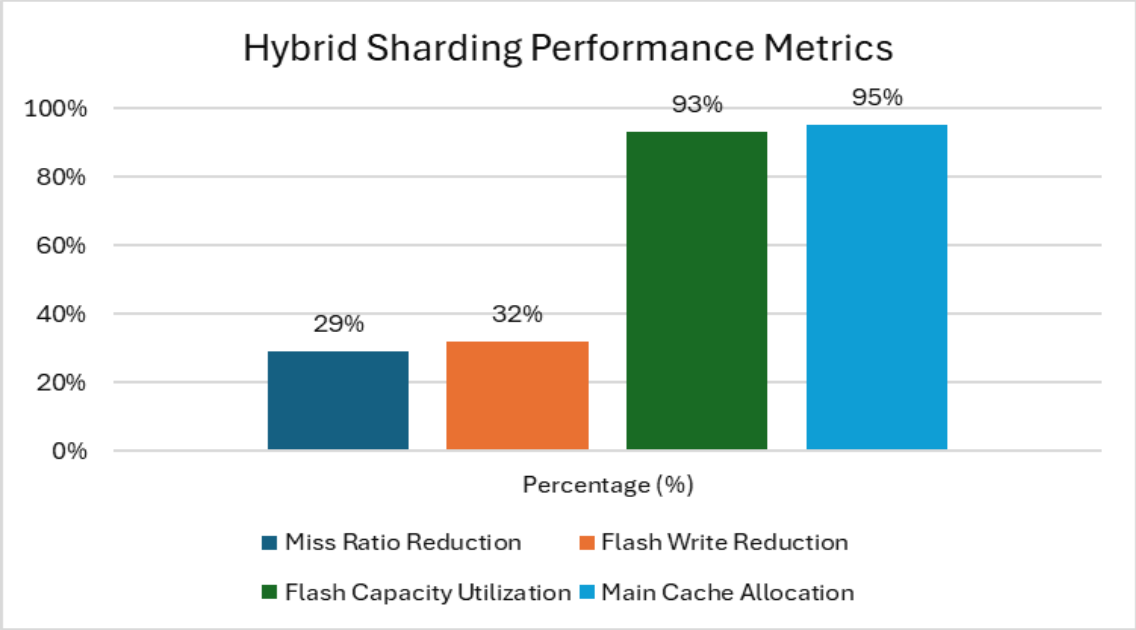

Figure 1: *Hybrid Sharding Performance Improvements and Resource Distribution [8]*

Table 1*: Switch Buffer Configuration Parameters [5, 6]*

Buffer Parameter	Value	Purpose
Headroom Buffer	22.4 KB per port per priority	Accommodate in-flight packets
Priority-based Flow Control Threshold	24.47 KB (maximum)	Flow control activation point
Dynamic Threshold Beta Parameter	8	Adaptive threshold adjustment
ECN Threshold	< 21.75 KB	Congestion notification trigger
ECN Minimum Threshold	5 KB	Lower bound for marking
ECN Maximum Threshold	200 KB	Upper bound for marking
Maximum Marking Probability	1%	Gentle marking slope for fairness
Alpha Update Parameter (g)	1/256	Rate control stability

Table 2: *DRAM Overhead Breakdown for Sharding Architecture [7]*

Component	Naive Design	Kangaroo Design
Flash Offset	29 bits	19 bits
Tag Size	29 bits	9 bits
Next-Pointer	64 bits	16 bits

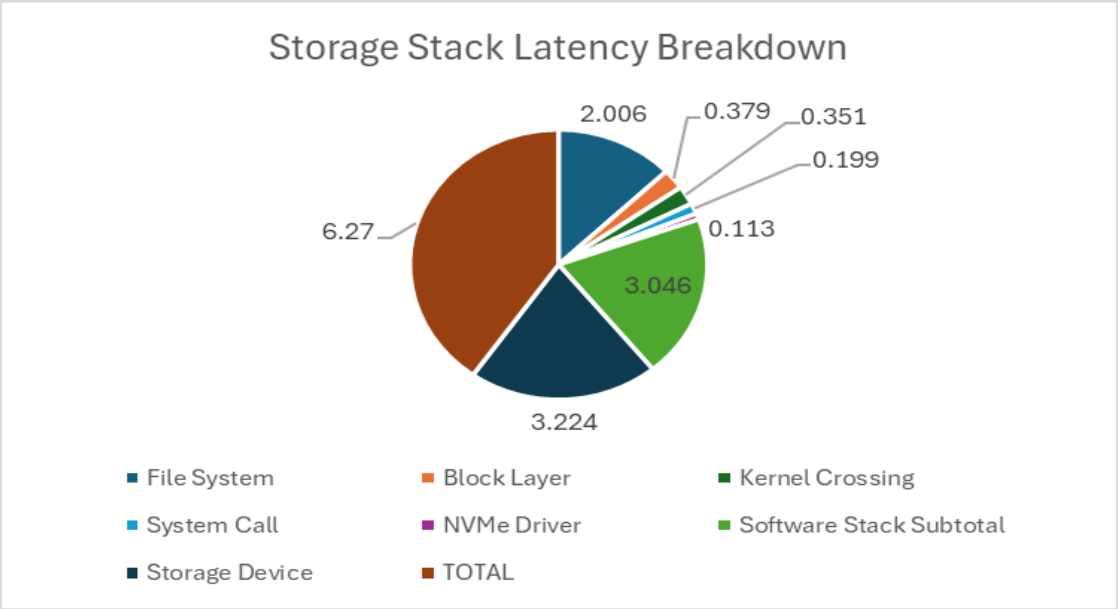

Figure 2: *Storage Stack Latency Breakdown for 512-byte Random Reads [9, 10]*

8. Conclusions

By using the above infrastructure, the developer velocity from the integration of these infrastructures into both production and load test environments has increased dramatically. Engineers now launch experiments and immediately receive percentile profiles matching production formats. This productionized approach to tail-latency diagnostics catches performance regressions before user impact. The system proves that sophisticated observability need not compromise efficiency. At scale, innovation ability directly correlates with observation capability of the most difficult fraction of traffic. The hybrid sharding and real-time thresholding architecture demonstrates this principle effectively. Design provides right surgical visualization, where it is needed most. Centralized controllers are used for consistent data through global consistency; sharded persistence enables scalable approach to horizontal scaling. Minimal reporting allows low overhead for standard transaction types while conditional profiling will help collect all diagnostic material for unexpected requests. Design philosophy based on outcomes is critical to accurate tail-latency measurements. Classic uniform sampling methods are unable to achieve equivalent diagnostic depth due to excessive resource costs. The ability to uniquely identify each provided traffic slice helps to improve methodologies of measuring the performance across different types of user populations. Load-test facilities close the gap between testing and live production environments enabling engineers to be more confident about the performance characteristics of their code before it gets put into production deployment. The system follows modern observability principles in which instrumentation dynamically adapts to the observed behavior of each component.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

1. Marios Kogias and Edouard Bugnion, "Tail-tolerance as a Systems Principle not a Metric," ACM Digital Library, 2020. Available: https://dl.acm.org/doi/10.1145/3411029.3411032
2. Mohammad Alizadeh, et al., "Less is more: trading a little bandwidth for ultra-low latency in the data

center," ACM Digital Library, 2012. Available: https://dl.acm.org/doi/10.5555/2228298.2228324

3. Ravi Netravali, et al., "Vesper: Measuring Time-to-Interactivity for Modern Web Pages," USENIX, 2018. Available: https://www.cs.princeton.edu/~ravian/publications/vesper_nsdi18.pdf
4. Kevin Hsieh, et al., "Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds," USENIX, 2017. Available: https://www.usenix.org/sites/default/files/conference/protected-files/nsdi17_slides_hsieh.pdf
5. Ahmed Saeed, et al., "Eiffel: efficient and flexible software packet scheduling," ACM Digital Library, 2019. Available: https://dl.acm.org/doi/10.5555/3323234.3323237
6. Yibo Zhu, et al., "Congestion Control for Large-Scale RDMA Deployments," ACM SIGCOMM Computer Communication Review, 2015. Available: https://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p523.pdf
7. Sara McAllister, et al., "Kangaroo: Caching Billions of Tiny Objects on Flash," Proceedings of SOSP, 2021. Available: https://www.pdl.cmu.edu/PDL-FTP/NVM/McAllister-SOSP21.pdf
8. Patrick Stuedi, et al., "Crail: A high-performance I/O architecture for the Apache data processing ecosystem," OpenFabrics Alliance, 2017. Available: https://www.openfabrics.org/images/eventpresos/2017presentations/109_Crail_BMetzler.pdf
9. Behnam Montazeri, et al., "Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities," ACM SIGCOMM Computer Communication Review, 2018. Available: https://people.csail.mit.edu/alizadeh/papers/homa-sigcomm18.pdf
10. Yuhong Zhong, et al., "XRP: In-Kernel Storage Functions with eBPF," USENIX Association, 2022. Available: https://www.usenix.org/system/files/osdi22-zhong_1.pdf