Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

High-Precision Load Balancing: Solving Tail Latency via State-Aware Probing and Split-Task Methodology

Varun Raj
Independent Researcher, USA

ARTICLE INFO

Received: 05 March 2026

Accepted: 08 March 2026

ABSTRACT

One of the biggest challenges with hyper-scale distributed systems is routing requests across thousands of microservices. Customary load-balancing approaches are often a bottleneck for ultra-low tail latency systems because of caches holding stale metadata, such as routes, which can be a problem in high-QPS systems. The server state is propagated with a time lag with respect to the load signal, leading to information asymmetry bottlenecks and hotspots, i.e, delays in the server responsiveness. State-aware probing measures these effects by accurately tracking the most recent states of the server task internals using synchronous piggybacked telemetry and asynchronous probe intervals. This probing uses queue depth, CPU, and memory pressure to create cluster maps to route requests. The current capacity (not historic averages) of these nodes is then used to route a request to the appropriate nodes that can process it. Recursive load-balancing policies using split-task experimental designs, which isolate and measure each variable with extreme precision, change the nature of load balancing from static algorithmic spreading to dynamic information-rich orchestration. This combination greatly increases infrastructure efficiency, simplifying the debugging and optimization of critical workloads such as machine learning inference and real-time search indexing.

Keywords: Tail Latency Optimization, State-Aware Load Balancing, Distributed Systems Orchestration, Real-Time Capacity Mapping, Split-Task Experimentation

1. Introduction

As distributed systems grow in size and begin to take on hyper-scale service architectures, the orchestration of requests across thousands of microservices becomes the most critical aspect of the stability and performance of these systems. Normal load-balancing techniques, which were sufficient to handle moderate amounts of traffic, have become the bottleneck for ultra-low tail latency distributed systems.

New approaches are emerging that have begun to move away from blind distribution systems treating all the servers in the same way, and towards state-aware orchestration mechanisms that are responsive to the state of the infrastructure in real-time. These combine system telemetry with experimentation at a fine level, providing a high degree of dynamic adaptability.

The importance of scheduling perspectives in reducing tail latency has been recognized in key-value stores and distributed databases because request completion time on heterogeneous pools of servers is highly variable [1]. Even small variations in processing time tend to increase as they move through several tiers of services and come to have a large impact on tail latency. Many customary load balancing algorithms, such as round-robin and least-connections, do not take into account such dynamic variation in server capacity and workload characteristics.

Congestion-aware mechanisms are yet another evolution of load-balancing technology, as network congestion in datacenter fabrics adds latency that customary load balancers are both unaware of and unable to alleviate [2]. In virtual edge environments, load balancers take into consideration both the

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

server and the network path load in order to make better end-to-end latency routing decisions, rather than just relying on the server load.

A recent survey works on load balancing at extreme scale in distributed systems, focusing on techniques that reduce tail latency by (1) state-aware probing and (2) systematic experimentation. This results in a fundamental rethinking of the way distributed traffic loads are managed every millisecond.

2. Limitations of Existing Load-Balancing Techniques

Many conventional load-balancing algorithms and architectures are ineffective at hyper-scale. Weighted round-robin and least-connections are particularly susceptible to the stale metadata problem. This problem is aggravated in high-query-per-second systems such as distributed microservice architectures.

The ratio of server task replicas to client instances causes a scaling bottleneck. Since there is no central balancer that keeps an accurate view of the server load in real-time across thousands of endpoints, the state of the server may have changed considerably by the time that a load signal reaches the client. Thousands of requests could be processed in a few milliseconds during this information propagation delay. Measurement studies indicate that the majority of congestion events in datacenters last less than three microseconds [4]. Microbursts responsible for over ninety percent of packet loss exhibit durations below this threshold. Traditional load balancers with control loops requiring tens to hundreds of microseconds cannot react effectively to these transient congestion episodes.

While adaptive load balancing has been well established for the inference of deep neural networks in collaborative edge computing platforms [3], the heterogeneous nature of edge computing devices presents significant challenges. Three representative edge platforms demonstrate this heterogeneity clearly. The Odroid XU4 operates with a thermal design power of eight watts. The Raspberry Pi4 functions at nine watts thermal design power. The Jetson Nano requires ten watts for its operations [3]. These devices exhibit vastly different computational capabilities despite targeting similar edge deployment scenarios. Load balancers must consider the inference accuracy requirements and acceptable latency of their edge infrastructure when determining task allocations across such heterogeneous resources.

Additionally, there are accuracy-aware workloads that cannot be served by the best-performing node since some edge nodes may only deliver low accuracy with higher processing speed. Experimental evaluations using MobileNetV2 models across heterogeneous edge devices reveal this accuracy-performance trade-off explicitly [3]. Six different versions of MobileNetV2 models were tested with multiplication width parameters of 0.35, 0.5, 0.75, 1.0, 1.3, and 1.4. These models exhibited accuracy ranging from 92.5 percent to 82.9 percent. The Jetson Nano consistently outperformed both Raspberry Pi4 and Odroid XU4 across different model configurations. However, both Raspberry Pi4 and Odroid XU4 could provide similar performance to Jetson Nano by accepting lower accuracy thresholds. Standard load-balancing metrics such as average task execution time do not account for such characteristics. This creates a complex exploration space where workload partitioning must jointly consider node-level heterogeneity and application-specific accuracy requirements.

Micro-level load balancing presents an additional challenge, particularly in datacenters, where decisions are made on the order of milliseconds [4]. Packet-level routing decisions must consider flow characteristics and current traffic at a node, or queue state. Experiments with switches of varying sizes, including thirty-two-port, forty-eight-port, and two hundred fifty-six-port configurations, demonstrate these timing constraints [4]. Drill-style mechanisms distribute traffic over a large number of parallel paths in datacenter topologies. These mechanisms require the fabric state to be

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

rapidly available across microsecond timescales that centralized controllers cannot achieve. The distributed nature of load-balancing decisions across multiple input ports creates coordination challenges. Simulation results show that optimal performance occurs with two random samples per routing decision rather than exhaustive comparison across all output ports [4].

As a result of this asymmetry, many distributed systems have common failure modes. One such mode results when requests are routed based on stale load information, producing hotspots in the system. A typical cause is that routing decisions rely on previous state information, which is no longer accurate. Control loops operating at timescales of tens to hundreds of microseconds cannot respond to congestion events lasting only three microseconds [4]. As a result, tail latency spikes are unavoidable. Packet reordering becomes inevitable under such conditions, with measurements showing approximately 0.02 percent of packets delivered out of order even under optimized micro load-balancing schemes [4]. This represents two packets in every ten thousand requiring reordering buffers.

These issues become more complex at scale, where the existing routing algorithms do not have the required scale-latency granularity to operate on geographically distributed resources. Experimental validation using Clos network topologies with four to six leaf switches, six spine switches, and sixteen hosts attached to each leaf switch reveals these scalability challenges [4]. With increased load and diversity, these assumptions are not valid. Uniform workload distribution strategies fail to meet performance requirements under heterogeneous conditions, exhibiting average performance violations of 41.52 percent [3]. Asymmetric distribution approaches improve resource utilization but remain limited by rated device capabilities. Uniform distribution with aggressive approximation achieves performance targets but violates accuracy requirements by 5.2 percent on average [3]. The need for more advanced systems to meet the growing demands of systems at scale becomes critical. Table 1 presents a comprehensive comparison of conventional load-balancing algorithms and their specific limitations in hyper-scale distributed environments, highlighting the challenges each approach faces in managing modern microservice architectures.

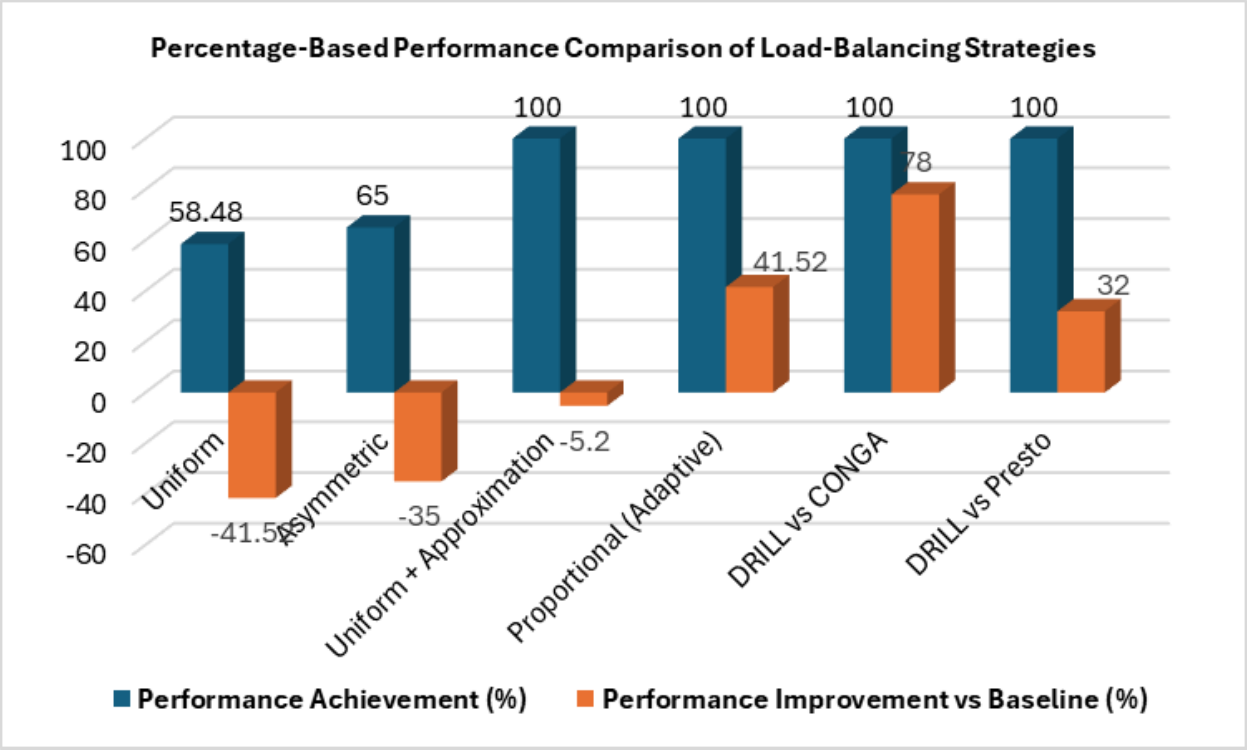

Fig. 1: Performance-Accuracy Trade-off Analysis in Conventional Load-Balancing Systems [3, 4]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3. The Necessity of High-Precision Experimental Methodology

These improvements often take the form of low-level architectural changes to the RPC stack. Small optimizations are introduced to service the long tail at extreme percentiles. However, marginal improvements to performance can often be subsumed by the sheer statistical variance created by running a multi-billion request cluster. The challenge intensifies when attempting to isolate specific performance gains from background noise in production environments.

In many of these cases, conventional A/B testing does not provide sufficient resolution for detecting subtle improvements. With millions of queries processed simultaneously, improvements at the highest percentiles are virtually impossible to separate statistically. The signal-to-noise ratio degrades significantly at extreme percentiles where tail latency optimization matters most. Without real-time control-loop frameworks to add isolation between variables, it is difficult for engineers to establish whether a policy is working or simply benefiting from a random fluctuation. Traditional statistical methods lack the granularity needed to confidently attribute performance changes to specific architectural modifications.

Predictive load balancing employs machine learning algorithms to forecast server load capacity and future workload patterns [5]. Three primary algorithmic approaches are typically compared in distributed systems evaluation. Round Robin provides deterministic baseline behavior with predictable distribution patterns across servers. Weighted Round Robin introduces static prioritization based on predetermined server capacities and resource allocations. Machine learning approaches enable dynamic adaptation by learning from historical traffic patterns and server response behaviors. Experimental evaluations in containerized environments with heterogeneous resources demonstrate distinct performance characteristics. Container configurations included systems with four CPU cores and twelve gigabytes of memory, three cores and nine gigabytes, and two cores and six gigabytes [5]. The predictive models analyze temporal patterns in request arrival rates and server processing times across these heterogeneous infrastructures.

Comparing fundamentally different algorithmic approaches with different workload characteristics presents significant experimental challenges. Round Robin has the benefit of providing a stable and deterministic baseline to which all optimization efforts can be compared. This deterministic behavior simplifies performance attribution during experimental validation phases. Weighted variants use heuristics that are helpful only in certain scenarios where server capacity differences are known in advance. The heuristics may perform well under stable conditions but degrade when workload characteristics shift unexpectedly. Controlled experiments across twenty thousand request samples revealed that Weighted Round Robin achieved an average latency of 198.90 milliseconds with a throughput of 2.07 million instructions per second [5]. Round Robin exhibited slightly higher latency at 214.67 milliseconds with a throughput of 1.92 million instructions per second. The machine learning-based predictive approach demonstrated 249.63 milliseconds average latency and 1.65 million instructions per second throughput despite achieving model accuracy with R-squared values of 0.81 on test data and 0.84 on validation data [5]. The added complexity and overhead of predictive models must be justified by measurable latency improvements. The computational cost of running machine learning inference for routing decisions must not exceed the latency savings achieved through smarter distribution strategies.

State management in cloud-native applications complicates the experimental validation of load-balancing techniques significantly [6]. Containerized microservices maintain an internal state that directly influences request processing latency. The effects of that state on latency are not always apparent through external observation alone. Application state can include cached data, active database connections, session information, and in-memory computational results. Load balancers

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

must take session affinity requirements into account when routing requests to maintain user experience consistency across interactions. Data locality considerations affect cache hit rates and data access latency across distributed storage systems. Experimental topologies with six hosts demonstrate these effects when deploying twelve network function instances across the infrastructure [6]. Stateful operations require careful coordination to prevent data inconsistency across replicas during concurrent access scenarios.

Container orchestration platforms must balance multiple competing objectives simultaneously during operation. Resource efficiency drives higher container density per physical host to maximize infrastructure utilization. Application performance guarantees require sufficient resource isolation between co-located workloads to prevent interference. State persistence mechanisms introduce input-output overhead that varies based on storage backend characteristics and network topology [6]. Network delay distributions in datacenter fabrics show mean values of 367 microseconds between host pairs for typical deployments [6]. Experiments that do not consider these state dependencies may lead to misleading conclusions about system performance. Results from stateless test workloads often fail to generalize to production systems handling stateful operations with session persistence requirements. The experimental design must account for state transfer costs, warm-up periods, and cache population effects that influence steady-state performance characteristics. Access pattern variations, including write-heavy scenarios with fifty-seven percent write operations, eighteen percent read operations, and twenty-five percent combined operations, versus read-heavy patterns with fifty-seven percent read operations, eighteen percent write operations, and twenty-five percent combined operations, produce dramatically different performance profiles [6].

With this measurement gap, teams have little incentive to experiment with optimization strategies for distributed systems. They cannot perform controlled experiments to assess the success of a given optimization strategy with sufficient statistical confidence. When a large discrepancy exists between production deployment behavior and optimization goals, user experience is heavily impacted through increased latency and reduced throughput. Service-level objective violations cascade across dependent microservices in complex ways that are difficult to diagnose. This requires sensitive experimental platforms that are robust to small gains and capable of detecting subtle improvements with high statistical power. The platforms must provide reproducible environments that isolate the impact of specific changes from environmental variations caused by network congestion or resource contention.

By providing the ability to measure at high-fidelity, engineers can trust the measurements and quickly iterate forward when architectural changes are made. Statistical confidence intervals narrow significantly with improved measurement precision enabled by controlled experimental conditions. Engineers can detect smaller effect sizes and make data-driven deployment decisions based on quantitative evidence. Experimental frameworks processing one thousand state instances across six-host clusters with controlled workload patterns provide sufficient granularity for performance analysis [6]. Ultra-scale companies gain a competitive advantage through precise measurement of minute improvements that accumulate over time. They can optimize systems at granularities previously impossible for the industry to achieve with conventional testing methodologies. Incremental gains accumulate across millions of servers to produce substantial efficiency improvements in aggregate resource utilization. The measurement infrastructure becomes a strategic asset enabling continuous optimization at massive scale across geographically distributed deployments [5][6].

Table 1 presents empirical performance metrics obtained from controlled experiments comparing traditional and machine learning-based load-balancing strategies. All experiments were conducted on heterogeneous containerized infrastructure with varying computational resources. The data represents aggregate measurements across twenty thousand request samples per strategy using synthetic workloads with task sizes ranging from two hundred fifty to seven hundred fifty million instructions distributed according to log-normal probability distributions.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Load Balancing Strategy	Average Latency (ms)	Throughput (MIPS)	Container Resource Configuration	Experimental Sample Size	Model Accuracy (R^2)
Weighted Round Robin	198.9	2.07	4 cores/12GB, 3 cores/9GB, 2 cores/6GB	20000	N/A
Round Robin	214.67	1.92	4 cores/12GB, 3 cores/9GB, 2 cores/6GB	20000	N/A
Predictive ML (Runtime Features)	249.63	1.65	4 cores/12GB, 3 cores/9GB, 2 cores/6GB	20000	0.81
Predictive ML (CPU/Memory Features)	Higher than 249.63	Lower than 1.65	4 cores/12GB, 3 cores/9GB, 2 cores/6GB	20000	0.6

Table 1: Experimental Validation Metrics for Predictive Load-Balancing Techniques in Cloud-Native Microservices [5, 6]

4. Recursive Load Balancing and Fine-Grained Experimentation

In this new framework, principled decisions on architecture and experimentation close the measurement gap. The recursive load-balancing policy implicitly separates the client and server into mirrored replicas for comparison. By enabling integration into existing experimental infrastructures, this avoids having to replace them entirely.

The system enables side-by-side studies of routing policies using identical traffic and workload characteristics. A subset of clients is pinned to certain halves of servers in controlled experiments to eliminate crosstalk. The split-task technique produces percentiles with finer granularity than can be achieved with conventional techniques, enabling engineers to isolate and identify the source of latency improvements before wider deployment.

In practice, large-scale distributed systems testing also runs into corner cases and failure scenarios that theoretical models have not anticipated and that do not show up in laboratory testing [7]. Testing frameworks must be able to scale to thousands of nodes and achieve reproducibility and deterministic run behavior. However, the practicalities of deploying experimental infrastructure into production environments limit what testing techniques are viable.

Reproducibility is more difficult for experiments on distributed systems, which are made up of components that communicate via a network, and therefore changes in network latency or CPU scheduling can lead to different results. Therefore, testing frameworks have to account for the potential of non-determinism in distributed systems. The tension between realism and control underlies much of the field of distributed systems validation.

Model-agnostic predictive load balancers, which are based on ML predictions, introduce an additional layer of complexity [8]. Their performance must be evaluated for various workload patterns and traffic conditions to test generalization. However, a model tuned to a particular workload may perform poorly if the traffic patterns are different. Accordingly, in order to capture unexpected variations, experimental frameworks should cover diverse workloads.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

To eliminate confounding, keep the same traffic pattern for the two experimental conditions; the server hardware between the control and the treatment group is the same, and the environment is also the same for both experimental conditions. This ensures that the only difference that could influence the outcome is the routing policy.

This approaches the rigor of the scientific method, as teams can statistically prove that their algorithms are effective in extremely long tail scenarios where scrutiny is high. This lowers the risk of the changes made to production, as feedback on each iteration comes sooner, making the process of optimization clearer since problems are detected earlier. Table 2 delineates the critical components and methodologies employed in recursive load-balancing frameworks designed to enable fine-grained experimentation and precise performance attribution in distributed systems.

Framework Component	Implementation Strategy	Experimental Benefit
Client-Server Replica Isolation	Logical separation into mirrored replicas for comparison	Side-by-side routing policy evaluation under identical conditions
Split-Task Pinning Mechanism	Specific client subsets pinned to designated server halves	Elimination of crosstalk and confounding variables
Traffic Pattern Control	Identical workload characteristics across experimental groups	Isolation of the routing policy impact from environmental fluctuations
Hardware Matching Protocol	Control and treatment groups use equivalent server specifications	Attribution of performance differences solely to algorithmic changes
Workload Diversity Coverage	Testing across various traffic patterns and load conditions	Validation of model generalization beyond specific scenarios

Table 2: Components of Recursive Load-Balancing Experimental Framework [7, 8]

5. State-Aware Probing: Real-Time Capacity Mapping

The major technical contribution is the use of advanced probing to provide an up-to-date view of the state of every task in the distributed cluster. This consists of synchronous piggybacked telemetry combined with asynchronous probes sent at regular intervals between the synchronous piggybacks. The probing mechanism operates continuously to capture dynamic infrastructure conditions as they evolve in real time.

The probes may combine multiple metrics to create a more complete view of server health across distributed environments. Current queue depth provides insight into backlog and processing bottlenecks affecting request throughput. CPU availability indicates immediate and near-term processing capacity for incoming workloads. Memory pressure indicates resource contention that could bottleneck memory-intensive tasks if the demand grows. Experimental deployments across six hosts demonstrate how these metrics create a high-fidelity map of the cluster's real-time operational state [9]. Network delay measurements between host pairs show mean values of 367 microseconds in typical datacenter fabrics. The aggregation of telemetry data enables predictive routing decisions before performance degradation becomes visible to end users.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Quality of experience considerations guide load-balanced groupings of container schedulers in modern cloud deployments [9]. In co-located cloud environments, a trade-off must be made between resource efficiency and application performance guarantees. Container positioning affects resource consumption and end-user quality of experience in terms of latency and responsiveness. QoE-aware scheduling seeks to avoid resource contention between co-located containers that may otherwise degrade application performance. Experimental deployments with twelve network function instances distributed across six-host clusters demonstrate that container placement strategies significantly impact user-perceived quality metrics [9]. The scheduling algorithms must balance infrastructure utilization targets with strict quality requirements.

A challenge is mapping low-level resource measurements to quality degradation as perceived by the user. Different categories of applications exhibit different behaviors when their CPUs are throttled based on the workloads they process. Memory pressure affects cache-heavy workloads much more than stateless request processors that maintain minimal internal state. Network bandwidth contention between containers can cause latency spikes that standard schedulers cannot anticipate or avoid. State placement experiments involving one thousand data instances reveal the complexity of correlating infrastructure metrics with application performance [9]. Quality-of-experience models translate resource constraints into predicted user experience scores. These models enable proactive scheduling decisions before quality degradation occurs.

Container co-location scenarios introduce additional complexity in resource management and performance prediction. When multiple containers share the same physical host, resource interference becomes inevitable. Experimental results show that cluster loads between forty and fifty percent represent light conditions, while loads between fifty and sixty percent characterize heavy operational scenarios [9]. CPU throttling affects application response times in proportion to computational intensity. Memory contention forces containers to access slower storage tiers when primary memory fills. Network interface saturation creates queuing delays for outbound traffic from all co-located containers. The scheduler must predict these interference patterns and make placement decisions accordingly. State access waiting times stabilize between 2.5 and 2.75 milliseconds under optimized placement strategies [9]. Intelligent co-location reduces quality degradation compared to random placement strategies.

Access pattern characteristics significantly influence performance outcomes in containerized environments. Write-heavy workloads with eighteen percent read operations, fifty-seven percent write operations, and twenty-five percent combined operations exhibit different resource consumption patterns than read-heavy scenarios [9]. Read-heavy patterns comprising fifty-seven percent read operations, eighteen percent write operations, and twenty-five percent combined operations demonstrate lower network traffic overhead. Balanced patterns with thirty-three percent each of read, write, and combined operations provide intermediate performance characteristics. The replication strategy impacts network load substantially, with naive replication-everywhere approaches generating 1.7 times more traffic than optimized dynamic replication schemes [9]. These variations necessitate adaptive probing strategies that adjust monitoring intensity based on observed access patterns.

Dynamic low-latency models dynamically and incrementally adjust load balancing according to environmental changes in hybrid architectures [10]. Massively multiplayer online games represent applications with strict low-latency requirements where millisecond delays severely impact user experience. Player interactions require real-time synchronization across distributed game servers. Hybrid fog and edge architectures distribute game state and load across different tiers of infrastructure with varying capabilities. Load balancers route player interactions to minimize latency while maintaining game state consistency [10]. The architecture combines centralized cloud resources with edge nodes positioned near player populations.

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

State placement optimization becomes computationally intensive as system scale increases. Small-scale deployments with ten state instances complete placement calculations in 0.007 seconds using heuristic algorithms [9]. Medium-scale scenarios with one thousand states require 6.31 seconds for placement computation. Large-scale environments with five thousand states extend processing time to 122 seconds. Ultra-large deployments with ten thousand states demand 566 seconds for complete placement optimization [9]. These timing constraints necessitate efficient heuristic approaches rather than exhaustive optimization techniques. Optimal solvers exceed one-hour time limits for deployments beyond five thousand states under heavy cluster loads.

This allows the load balancer to make decisions that take the current state of the system into account in real time. Load is directed towards the nodes with the best characteristics at the right moment for each request type. The system self-adjusts to changes in workload patterns and cluster conditions at faster time scales than customary monitoring and control loops. Twenty thousand labeled request samples collected under controlled conditions provide sufficient training data for machine learning-based predictive models [9]. Proactive routing prevents hotspot formation rather than reacting after performance degradation occurs. The feedback mechanisms enable continuous optimization as infrastructure conditions evolve.

State-aware routing alleviates the information asymmetry problem by making routing decisions based on the current network and server state. Decisions reflect conditions from milliseconds rather than server state from seconds or minutes earlier, as in customary static load balancing schemes. This temporal accuracy greatly improves resource utilization and tail latency across percentiles. Experimental validation across fat-tree datacenter topologies with varying host counts demonstrates consistent performance improvements [9]. Hotspots become rare because the system preemptively avoids nodes that are approaching saturation thresholds. The probing overhead remains minimal compared to the performance gains achieved through informed routing. Figure 2 categorizes the telemetry metrics and decision parameters utilized in state-aware probing mechanisms to maintain real-time cluster capacity mapping and optimize routing decisions in distributed systems.

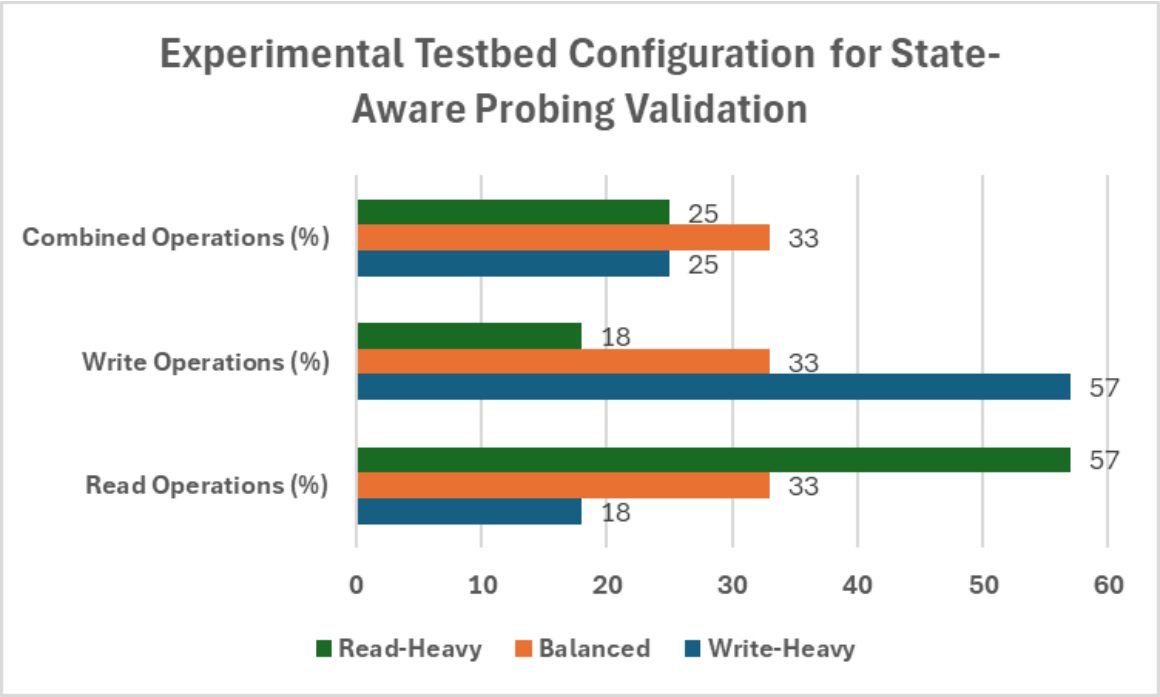

Fig. 2: State-Aware Probing Performance Metrics in Containerized Cloud Environments [9]

Journal of Information Systems Engineering and Management
2026, 11(2s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Conclusion

When combined with state-aware probing schemes and high-accuracy measurement systems, these ideas provide system designers with the ability to implement a range of customization in load balancing decisions to support performance-sensitive distributed workloads in hyper-scale systems, especially where tail latency is critical to user experience and reliability, such as in large-scale machine learning inference and real-time indexing for search, which require predictable performance at high percentiles. Implementing these capabilities into the RPC stack creates a more efficient and debuggable infrastructure. It will also give operators access to the routing decisions on a per-message basis, which at last provides levers so routing can move from guesswork to an informed data-driven decision process. The success of state-aware probing shows that load balancing in a dynamic system is a sensing problem, rather than an algorithmic one: static algorithms cannot compete with online systems in a distributed cluster. Organizations that can adopt these techniques will be much better equipped to confront the next generation of scale challenges without having to continually expand the cluster to accommodate the explosion of traffic and complexity. Future work can apply these ideas to other, more complicated distributed systems, utilizing more sources of telemetry data, and more advanced prediction models to reduce tail latency and improve the robustness of deploying infrastructure in environments that are heterogeneous.

References

[1] Sonia Ben Mokhtar, et al., "Taming Tail Latency in Key-Value Stores: A Scheduling Perspective," Euro-Par 2021: Parallel Processing - Springer Nature, 2021. Available: https://link.springer.com/chapter/10.1007/978-3-030-85665-6_9

[2] Naga Katta, et al., "Clove: Congestion-Aware Load Balancing at the Virtual Edge," ACM Digital Library, 2017. Available: https://dl.acm.org/doi/10.1145/3143361.3143401

[3] Zain Taufique, et al., "Adaptive Workload Distribution for Accuracy-aware DNN Inference on Collaborative Edge Platforms," arXiv, 2023. Available: https://arxiv.org/pdf/2310.10157

[4] Soudeh Ghorbani, et al., "Micro Load Balancing in Data Centers with DRILL", ACM, 2015. Available: https://conferences.sigcomm.org/hotnets/2015/papers/ghorbani.pdf

[5] Elshan Rahimov, et al., "Predictive Load Balancing in Distributed Systems: A Comparative Study of Round Robin, Weighted Round Robin, and a Machine Learning Approach," Engineering Proceedings, 2026. Available: https://www.mdpi.com/2673-4591/122/1/26

[6] Márk Szalay, et al., "State Management for Cloud-Native Applications," Electronics 2021. Available: https://www.mdpi.com/2079-9292/10/4/423

[7] Nuno Machado, et al., "Minha: Large-Scale Distributed Systems Testing Made Practical," OPODIS 2019 - ACM, 2019. Available: https://drops.dagstuhl.de/storage/00lipics/lipics-vol153-opodis2019/LIPIcs.OPODIS.2019.11/LIPIcs.OPODIS.2019.11.pdf

[8] Snehal Chaflekar, et al., "Model Agnostic Predictive Smart Load Balancer in Microservices Environment," IEEE Xplore, 2024. Available: https://ieeexplore.ieee.org/abstract/document/10842849

[9] Marcos Carvalho, et al., "QoE-Aware Container Scheduler for Co-located Cloud Environments," IFIP, 2021. Available: https://dl.ifip.org/db/conf/im/im2021/211580.pdf

[10] Ernesto José García Fernández de Castro, et al., "Dynamic Low-Latency Load Balancing Model to Improve Quality of Experience in a Hybrid Fog and Edge Architecture for Massively Multiplayer Online (MMO) Games", MDPI, 2025. Available: https://www.mdpi.com/2076-3417/15/12/6379