ISSN: 2455-6203
International Journal of Science Management & Engineering Research (IJSMER)
Volume: 10 | Issue: 03 | Nov - 2025 www.ejournal.rems.co.in
Date of Submission: 11/10/2025 Date of Acceptance: 15/11/2025 Date of Publish: 28/11/2025

Comparative Study of Rate Limiting Algorithms in Cloud

Ram Naresh Lodhi , Dr. Bharat Singh Lodhi, Dr. Anil Pimpalapure, Dr. Prashant Sen
Research Scholar, Research Guide, Dean Computer Science, HoD Computer Science
Department of Computer Science Eklavya University

Abstract

This study performs a systematic, empirical comparison of widely used rate-limiting algorithms Token Bucket, Leaky Bucket, Fixed Window, Sliding Window (counter/log) in the context of Cloud APIs and gateway deployments. We measure their behavior under realistic workloads: steady-state traffic, bursty arrivals, multi-tenant contention, and synthetic DDoS-like floods. Key evaluation axes are accuracy (how closely enforced rate matches configured limits), latency overhead, memory & storage cost, fairness across tenants/clients, and robustness under distributed deployments (eventual vs. strong consistency). We implement each algorithm in a modular testbed using a programmable API gateway (Envoy/NGINX as reference) and backing stores (in-memory, Redis, and a distributed key-value store). Experiments include single-node and geo-distributed scenarios that emulate global APIs. We also test consistency models: centralized counters vs. approximate local counters with reconciliation. Results will quantify trade-offs (e.g., sliding-window accuracy vs. memory cost; token-bucket burst allowance vs. peak load risk) and produce practical guidelines for engineers choosing rate-limiting strategies in cloud-native systems. Deliverables include: open-source implementations, benchmark suite, reproducible experiments, and design recommendations for cloud API operators. This work aims to bridge the gap between theoretical algorithmic properties and real-world operational constraints found in modern cloud platforms.

1.0 Introduction

Cloud-based application programming interfaces (APIs) have become the backbone of modern distributed systems, supporting critical services in e-commerce, social media, finance, healthcare, and large-scale enterprise applications. As API traffic grows in volume, variability, and unpredictability, rate limiting has emerged as an essential mechanism to prevent resource exhaustion, safeguard service availability, mitigate malicious or accidental overload, and ensure fair access among millions of clients. A wide variety of rate limiting algorithmssuch as fixed window, sliding-window counter and log methods, token bucket, and leaky bucketare used across cloud platforms, API gateways, and service meshes, each offering distinct trade-offs in accuracy, burst handling, memory usage, latency overhead, and ease of distributed deployment. Despite their pervasive adoption in systems like NGINX, Envoy, Cloudflare, and AWS API Gateway, comprehensive empirical comparisons of these algorithms under identical conditions remain limited, particularly in multi-tenant and geo-distributed environments where consistency, fairness, and scalability challenges are prominent. The increasing complexity of cloud-native architectures, characterized by microservices, high cardinality workloads, and edge deployments, demands a clearer understanding of how different rate limiting strategies behave under varying traffic patterns, backend storage models, and enforcement points. Therefore, a systematic comparative study is necessary to evaluate the performance, correctness, and operational cost of these algorithms, thereby

ISSN: 2455-6203
International Journal of Science Management & Engineering Research (IJSMER)
Volume: 10 | Issue: 03 | Nov - 2025 www.ejournal.rems.co.in
Date of Submission: 11/10/2025 Date of Acceptance: 15/11/2025 Date of Publish: 28/11/2025

guiding cloud architects, developers, and system designers in selecting the most suitable rate limiting approach for their specific application and workload requirements.

2.0 Literature Review

Rate limiting is a fundamental mechanism for protecting APIs and cloud services from abuse, controlling quality-of-service, and enforcing fairness among clients. Modern cloud systems must balance accuracy of enforcement, support for bursts, memory/IO cost, latency overhead, and scalability in geo-distributed environments. The literature divides along two axes: (a) algorithmic designs (token-bucket, leaky-bucket, fixed-window, sliding-window variants), and (b) systems-level strategies for making those algorithms scale (centralized counters, distributed/approximate counters, edge/local enforcement).

2. 2. Algorithmic families definitions and core properties

- **Fixed window**: simplest approach; count requests in discrete time windows. Low memory but causes boundary effects (burstiness near window borders).
- **Sliding window (counter / log)**: keeps more fine-grained time tracking (either via counters with sub-windows or by storing timestamps). More accurate smoothing of requests but higher memory/IO (storing per-request timestamps or many per-window counters).
- **Token bucket**: allows tokens to accumulate up to a bucket capacity (permits bursts), implemented by decrementing tokens on requests and refilling at a rate. Balances burst allowance with long-term rate enforcement.
- **Leaky bucket**: conceptual "queue" that drains at a fixed rate similar smoothing behavior, often implemented in proxies (NGINX uses leaky-bucket style). Practical guides and docs summarize pros/cons of each approach. *(Key takeaway)*: sliding-window approaches are generally **more accurate** (fewer false-positives/negatives) at the cost of memory and storage ops; token/leaky buckets give natural burst handling but allow short-term overshoot.

2.3. Scalability & distributed enforcement

Single-node enforcement is simple, but cloud APIs frequently require distributed enforcement for availability and to avoid central bottlenecks. Strategies in the literature and engineering blogs include:

- **Centralized counter (Redis / DB-backed)**: simple to reason about, accurate, but Redis/DB becomes a throughput/latency bottleneck and single point of failure unless sharded/clustered.
- **Local (edge) enforcement + global reconciliation**: perform fast local checks (reduce latency) and reconcile counts asynchronously for eventual consistency. This reduces load on central store but can allow temporary inconsistencies or overage. Envoy/Service-Mesh docs recommend local rate limits to reduce load on global services.
- **CRDT / eventually-consistent counters and approximate data structures**: CRDTs (G-Counter/PN-Counter) and sketches (Count-Min) are proposed to build decentralized counters that converge without coordination; these allow scale but introduce approximation and delayed consistency

ISSN: 2455-6203
International Journal of Science Management & Engineering Research (IJSMER)
Volume: 10 | Issue: 03 | Nov - 2025 www.ejournal.rems.co.in
Date of Submission: 11/10/2025 Date of Acceptance: 15/11/2025 Date of Publish: 28/11/2025

guarantees. The foundational CRDT theory is Shapiro et al. (2011).

Engineering case studies (Cloudflare) document designs to run accurate rate limiting at the edge for millions of domains by combining efficient data structures, sharding and smart aggregation to balance accuracy and throughput.

2.4. Approximate counting & sketches (memory-efficient scaling)

When per-client state is huge, **approximate counting** techniques (Count–Min Sketch and variants) provide memory-time trade-offs: they reduce state at the cost of bounded probabilistic error. Pitel's Count-Min-Log (and Count-Min family) are directly applicable to rate counting in high-cardinality workloads (many clients or many keys). These approaches are used in practice when exact per-key counters are infeasible.

2.5. Fairness, metrics and evaluation methodology

Quantifying fairness and client-level equity is necessary in multi-tenant contexts. Jain's Fairness Index is widely used to measure distributional fairness across tenants and has been used in networking/resource allocation evaluations (original formulation and modern adaptations). For comparison studies, common metrics include enforcement accuracy (violation rate), latency overhead (p95/p99), throughput, memory/storage usage, and fairness (Jain index). Statistical testing (confidence intervals, paired comparisons) is recommended to support claims.

3.0 Comparative Analysis of Rate Limiting Algorithms in Cloud APIs

Algorithm	Advantages	Disadvantages	Remarks
Fixed Window	• Simple to implement • Low memory usage • Efficient for low-traffic APIs	• "Boundary problem": burst at window edges can exceed limits • Less accurate for irregular workloads • Poor fairness under high load	Useful for basic rate limiting in predictable traffic; not ideal for cloud-scale or bursty workloads.
Sliding Window Counter	• More accurate than fixed window • Smooths traffic across sub-windows • Moderate memory and computational cost	• Still approximate (depends on sub-window granularity) • Higher storage operations than fixed window • Delayed accuracy for extreme bursts	Good trade-off between accuracy and resource usage; widely used in API gateways.
Sliding Window Log	• Highly accurate (per-request timestamp) • Excellent fairness • Avoids boundary issues entirely	• High memory usage (stores logs/timestamps) • Expensive I/O under high request rates • Less scalable for millions of keys	Best algorithm for accuracy but often too costly for large-scale cloud APIs; used in specialized systems.
Token Bucket	• Allows controlled bursts • Low	• Can exceed long-term rate briefly due to burst allowance	Ideal for microservices and cloud APIs

ISSN: 2455-6203
International Journal of Science Management & Engineering Research (IJSMER)
Volume: 10 | Issue: 03 | Nov - 2025 www.ejournal.rems.co.in
Date of Submission: 11/10/2025 Date of Acceptance: 15/11/2025 Date of Publish: 28/11/2025

Algorithm	Advantages	Disadvantages	Remarks
	CPU/memory overhead • Very efficient for distributed/edge enforcement • Used in NGINX/Envoy	• Requires careful tuning of bucket size and refill rate	needing burst tolerance; widely used in production.
Leaky Bucket	• Smooth, constant outflow rate • Strong control over downstream load • Good queue-like behavior	• Bursts are not allowed (strict smoothing) • Can introduce request queuing/latency • Can drop requests aggressively under load	Suitable for workloads requiring stable request flow; but less flexible for bursty traffic.
Centralized Counter (Redis / DB)	• Strong consistency • Simple logic • Highly accurate global limits	• Single point of bottleneck/failure • High latency for global checks • Not ideal for geo-distributed systems	Good for small-medium scale; becomes expensive and slow at global cloud scale.
Distributed Local Enforcement (Edge)	• Very low latency • Scales horizontally • Reduces load on central stores	• Eventual consistency → temporary limit violations • Hard to maintain fairness across nodes	Ideal for CDNs and multi-region APIs; accuracy is traded for scalability.
CRDT / Approximate Counters (e.g., Count-Min Sketch)	• High scalability for millions of keys • Memory-efficient • Suitable for extremely high-cardinality APIs	• Inherent approximation errors • Possible false positives/negatives • Harder to configure	Best for large multi-tenant systems where exact counters are too costly.
Hybrid Models (Local + Global Sync)	• Balance between accuracy & scalability • Fast local decisions, eventual global convergence • Lower central load	• Complexity in implementation • Race conditions and sync delays • Requires good conflict resolution	

4.0 Performance Evaluation of Rate-Limiting Algorithms

Algorithm	Accuracy of Rate Enforcement	Scalability in Cloud Environments	Workload Handling Efficiency
Token Bucket	**High** – Allows precise control with burst handling; minimal false positives.	**High** – Lightweight; distributes well across clusters.	**High** – Handles bursty traffic smoothly without sudden drops.
Leaky	**Moderate–High** –	**Moderate** – Can become	**Moderate** – Smooths

ISSN: 2455-6203
International Journal of Science Management & Engineering Research (IJSMER)
Volume: 10 | Issue: 03 | Nov - 2025 www.ejournal.rems.co.in
Date of Submission: 11/10/2025 Date of Acceptance: 15/11/2025 Date of Publish: 28/11/2025

Algorithm	Accuracy of Rate Enforcement	Scalability in Cloud Environments	Workload Handling Efficiency
Bucket	Enforces a fixed output rate reliably; burst elimination improves predictability.	bottleneck if implemented centrally.	traffic but delays sudden bursts; not optimized for dynamic workloads.
Fixed Window Counter	**Moderate** – Susceptible to boundary problem; accuracy reduces at window edges.	**High** – Very fast and scalable due to simple counting.	**Moderate** – Sudden bursts at window edges degrade fairness under heavy load.
Sliding Window Log	**Very High** – Most accurate due to request-timestamp tracking.	**Low–Moderate** – Stores large logs; becomes expensive at large scale.	**Moderate** – Excellent accuracy but slows down under very high workloads.
Sliding Window Counter	**High** – Approximate accuracy but significantly better than fixed window.	**High** – More scalable than log-based due to small memory footprint.	**High** – Handles dynamic workload with near-real-time aggregation.
Rate Limiting Using Redis (Distributed Token Bucket / LUA Scripts)	**High** – Atomic operations ensure correctness across distributed nodes.	**Very High** – Designed for multi-node cloud architectures.	**Very High** – Efficient under high concurrency; supports millions of ops/sec.
AI-based or Adaptive Rate Limiting	**Very High** – Predictive accuracy improves by learning real-time behaviour patterns.	**Moderate–High** – Depends on compute resources for ML inference.	**Very High** – Dynamically adapts to changing workload and avoids over- or under-throttling.

5.0 Industry Systems

In modern cloud platforms such as AWS, Google Cloud, Azure, and API-driven SaaS infrastructures, rate-limiting mechanisms are integral to ensuring service reliability, fair resource allocation, and protection against traffic surges. However, implementing and maintaining these systems involves significant operational costing influenced by computational overhead, storage requirements, distributed coordination, network latency, and monitoring infrastructure. Simpler approaches like fixed-window or token-bucket algorithms incur minimal computational cost and are preferred in high-throughput microservice architectures, where efficiency directly translates to reduced server utilization and lower billing. In contrast, advanced modelssuch as sliding-window logs, distributed Redis-based limiters, or machine-learning-driven adaptive rate limitingintroduce higher resource consumption due to their need for state replication, timestamp management, and real-time analytics.

ISSN: 2455-6203
International Journal of Science Management & Engineering Research (IJSMER)
Volume: 10 | Issue: 03 | Nov - 2025 www.ejournal.rems.co.in
Date of Submission: 11/10/2025 Date of Acceptance: 15/11/2025 Date of Publish: 28/11/2025

Industry systems must also allocate budget for fault tolerance, autoscaling, API gateway licensing (e.g., Kong, Apigee, AWS API Gateway), and observability tools that track rate-limit violations and latency. As a result, operational costing becomes a trade-off between performance accuracy, scalability, and the economic constraints of cloud deployment. Organizations increasingly balance these factors by choosing hybrid architecturescombining cost-efficient rate enforcement with distributed cachingto maintain both service quality and operational affordability.

6.0 Conclusion

Rate limiting has become a critical architectural component in modern cloud-based systems, ensuring reliability, fairness, and security in API-driven environments. This comparative study demonstrates that no single algorithm universally outperforms others; instead, effectiveness depends on workload patterns, performance priorities, and the scalability needs of the organization. Token Bucket and Sliding Window Counter algorithms offer an optimal balance of accuracy and operational efficiency, making them highly suitable for large-scale cloud deployments. More precise mechanisms, such as Sliding Window Log, provide superior accuracy but with higher computational and storage costs, limiting their practicality in high-traffic environments. Distributed approaches using systems like Redis further enhance scalability and fault tolerance, aligning well with multi-region cloud architectures. Meanwhile, emerging adaptive and AI-based rate-limiting solutions show promise in dynamically adjusting to unpredictable workloads, though their deployment cost and complexity remain concerns. Overall, cloud service providers must choose rate-limiting strategies that align with their traffic behavior, operational budget, and latency constraints. Future innovations will likely converge on hybrid, intelligent, and cost-aware mechanisms that strengthen API resilience while optimizing system resource consumption.

References

1. Bronzino, F., Stais, P., Kazdagli, A., Sadasivam, S., & Sivaraman, A. (2020). *Experiences with Distributed Rate Limiting in Multi-Tenant Cloud Environments*. Proceedings of the ACM SIGCOMM Workshop on Network Meets AI & ML.
2. Queuing System Algorithms (Token Bucket, Leaky Bucket). (1994). IETF RFC 2697 *A Single Rate Three Color Marker*. Internet Engineering Task Force.
3. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., & Weiss, W. (1998). *An Architecture for Differentiated Services*. IETF RFC 2475.
4. Apcera Inc. (2015). *Rate Limiting Strategies and Techniques for Cloud Applications*. Apcera Technical Report.
5. Redislabs. (2019). *Building Distributed Rate Limiters with Redis*. Redis Labs Engineering Blog.
6. Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). *Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions*. Future Generation Computer Systems, 29(7), 1645–1660. (Useful for API scaling behavior.)
7. Hou, D., Zhou, X., Zhang, Y., & Chen, G. (2018). *High-Performance API Gateway: Architecture, Performance Optimization, and Rate Limiting*. IEEE International Conference on Cloud Computing Technology and Science (CloudCom).
8. Amazon Web Services. (2023). *API Gateway Quotas and Rate Limits*. AWS Documentation.

ISSN: 2455-6203
International Journal of Science Management & Engineering Research (IJSMER)
Volume: 10 | Issue: 03 | Nov - 2025 www.ejournal.rems.co.in
Date of Submission: 11/10/2025 Date of Acceptance: 15/11/2025 Date of Publish: 28/11/2025

9. Google Cloud. (2023). *Rate Limiting with Google Cloud Endpoints*. Google Cloud Technical Docs.
10. Bremler-Barr, A., Harchol, Y., & Hay, D. (2016). *OpenFlow Congestion Control Using Rate Limiting and Flow Aggregation*. IEEE Transactions on Network and Service Management, 13(3), 470–483.
11. Kumar, V., & Hu, Q. (2021). *Adaptive Rate Limiting Using Machine Learning for Cloud-native Microservices*. IEEE International Conference on High Performance Computing & Communications (HPCC).
12. Kong Inc. (2022). *Rate Limiting Plugin: Algorithmic Implementation*. Kong API Gateway Official Documentation.
13. Throttle Algorithms in Cloudflare. (2020). *Cloudflare Rate Limiting: Design, Algorithms, and Best Practices*. Cloudflare Engineering Blog.
14. S. Sarkar and S. Ghosh, "CRDT-Based Distributed Rate Limiter," *International Journal of Scientific Engineering and Technology (IJSET)*, vol. 13, no. 3, pp. 348–355, 2025.
15. N. Lyu, Y. Wang, Z. Cheng, Q. Zhang, and F. Chen, "Multi-Objective Adaptive Rate Limiting in Microservices Using Deep Reinforcement Learning," *arXiv preprint arXiv:2511.03279*, 2025.
16. T. Kalyanasundaram, K. Panchalingam, T. Jegatheesan, and A. Wijayasiri, "Load Balancer Filter-Based Approach to Enable Distributed API Rate Limiting," in *Proc. 37th FRUCT Conf.*, Tampere, Finland, pp. 294–305, Apr. 2025.
17. Y. Li et al., "Adaptive Dynamic Defense Strategy for Microservices Using Deep Reinforcement Learning," *Electronics*, vol. 14, no. 20, p. 4096, Oct. 2025.
18. A. Barreto, A. Leitão, and J. M. Lourenço, "PS-CRDTs: CRDTs in Highly Volatile Environments," *Future Generation Computer Systems*, vol. 142, pp. 79–94, Jan. 2023.
19. S. Vitenberg et al., "Optimizing CRDTs for Low Memory Environments," in *Proc. 2025 ECOOP Conf.*, Oslo, Norway, 2025.