Magnetar (SLaB 2.0)
Phase - 1

Key Findings

1. Round Robin is deterministic and matches theory very closely (pages 1–4).

2. Random routing has higher variance in short runs but converges to expectation for larger N (you showed 1000 vs 2000 runs).

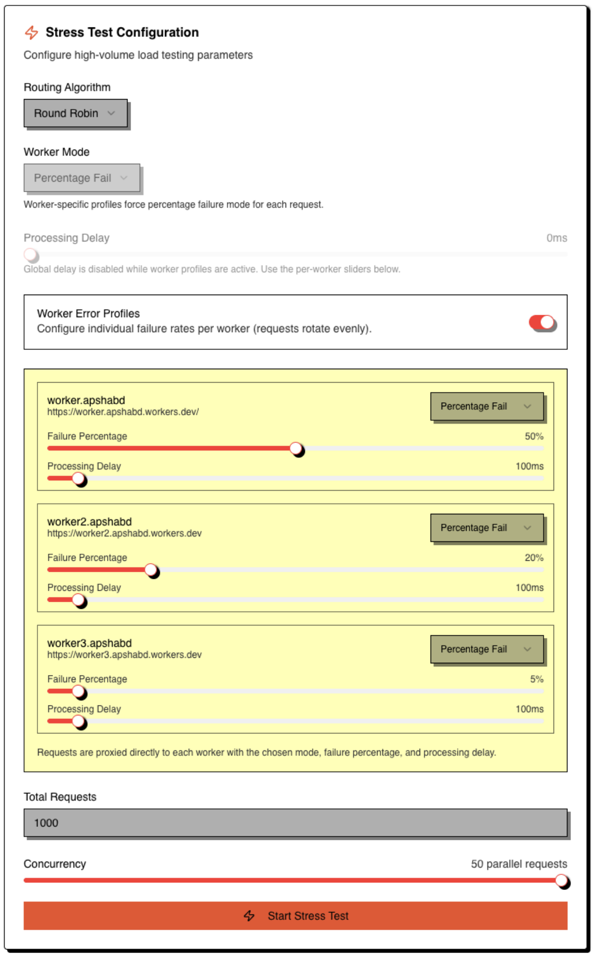
3. Throughput is limited by worker processing delay; routing overhead is negligible. (you measured ~13.6 req/s at 350ms delay)

4. Round Robin strictly exposes bad nodes (zero bias) — perfect as a baseline; this makes an excellent control group.

Sarvagya Kumar
25-26th November 2025

Configuration - 1

- **Conducted : Tuesday, 25th November 2025**
- **By : Sarvagya Kumar**



Stress Test Configuration

Configure high-volume load testing parameters

Routing Algorithm: Round Robin

Worker Mode: Percentage Fail

Worker-specific profiles force percentage failure mode for each request.

Processing Delay: 0ms

Global delay is disabled while worker profiles are active. Use the per-worker sliders below.

Worker Error Profiles

Configure individual failure rates per worker (requests rotate evenly).

worker.apshabd
https://worker.apshabd.workers.dev/
Percentage Fail
Failure Percentage: 50%
Processing Delay: 100ms

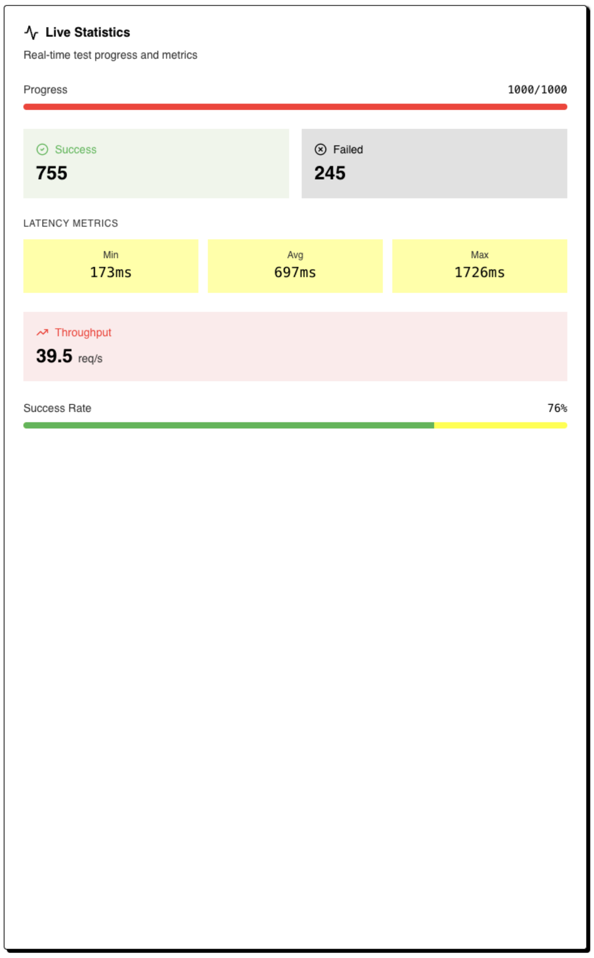
worker2.apshabd
https://worker2.apshabd.workers.dev
Percentage Fail
Failure Percentage: 20%
Processing Delay: 100ms

worker3.apshabd
https://worker3.apshabd.workers.dev
Percentage Fail
Failure Percentage: 5%
Processing Delay: 100ms

Requests are proxied directly to each worker with the chosen mode, failure percentage, and processing delay.

Total Requests: 1000

Concurrency: 50 parallel requests



Live Statistics

Real-time test progress and metrics

Progress: 1000/1000

Success: 755

Failed: 245

LATENCY METRICS

Min	Avg	Max
173ms	697ms	1726ms

Throughput: 39.5 req/s

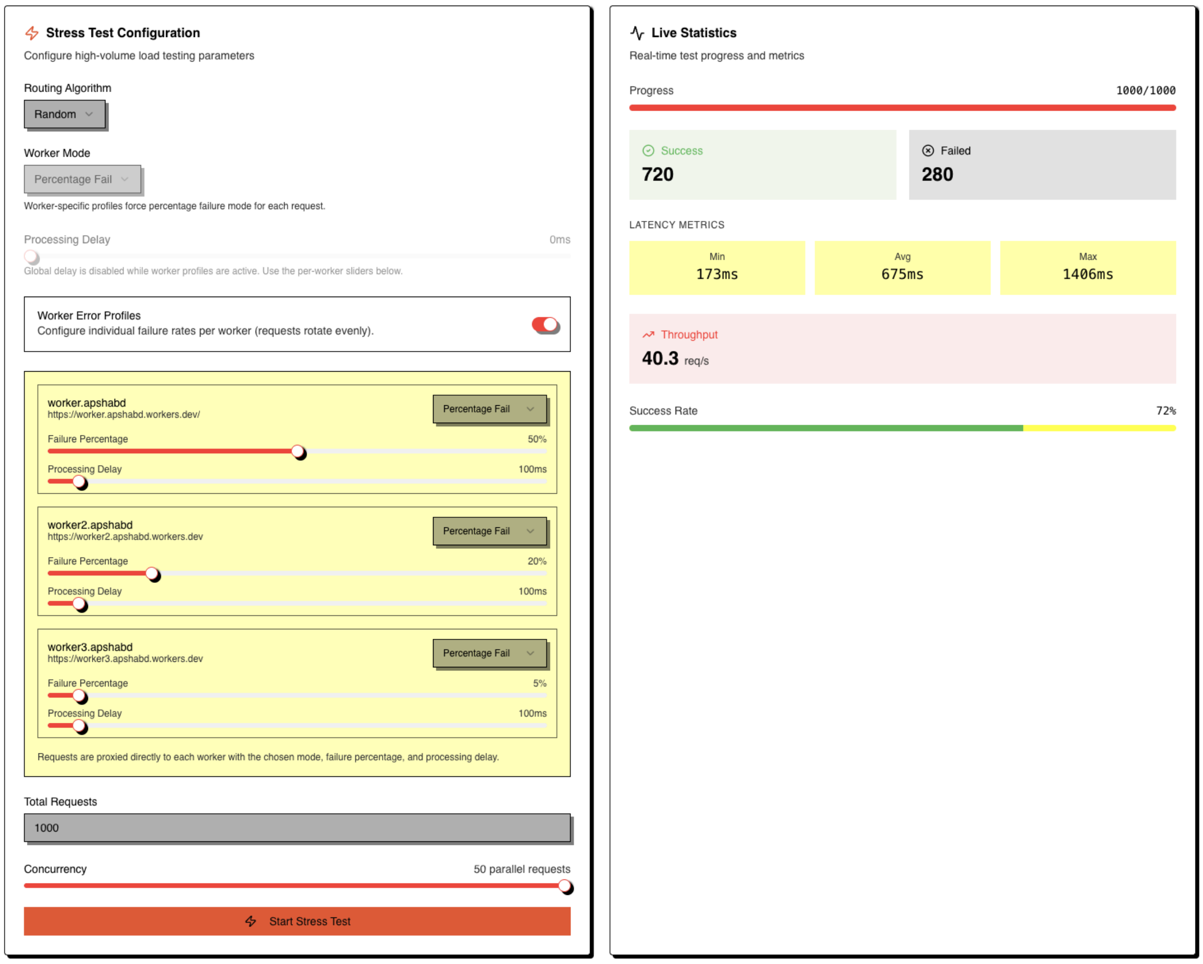
Success Rate: 76%

- This experimental observation captures the outcome of a **1000-request load test** executed at a concurrency level of **50 parallel requests**.
- The configuration utilizes a **Round Robin** routing algorithm to distribute traffic evenly across three distinct worker nodes, each assigned a specific "Percentage Fail" profile:
    - **Worker 1:** 50% Failure Rate
    - **Worker 2:** 20% Failure Rate
    - **Worker 3:** 5% Failure Rate
- **Results:** The system achieved a **76% Success Rate** (755 successes / 245 failures).

- This observed failure count (245) aligns closely with the theoretical expected failure rate of ~250 (the average of 50%, 20%, and 5% across three evenly distributed nodes).
- The test sustained a throughput of **39.5 req/s** with an average latency of **697ms**.

Configuration - 2

- **Conducted : Tuesday, 25th November 2025**
- **By : Sarvagya Kumar**



Stress Test Configuration
Configure high-volume load testing parameters

Routing Algorithm: Random

Worker Mode: Percentage Fail
Worker-specific profiles force percentage failure mode for each request.

Processing Delay: 0ms
Global delay is disabled while worker profiles are active. Use the per-worker sliders below.

Worker Error Profiles
Configure individual failure rates per worker (requests rotate evenly).

worker.apshabd
https://worker.apshabd.workers.dev/
Percentage Fail
Failure Percentage: 50%
Processing Delay: 100ms

worker2.apshabd
https://worker2.apshabd.workers.dev
Percentage Fail
Failure Percentage: 20%
Processing Delay: 100ms

worker3.apshabd
https://worker3.apshabd.workers.dev
Percentage Fail
Failure Percentage: 5%
Processing Delay: 100ms

Requests are proxied directly to each worker with the chosen mode, failure percentage, and processing delay.

Total Requests: 1000

Concurrency: 50 parallel requests

Live Statistics
Real-time test progress and metrics

Progress: 1000/1000

Success: 720
Failed: 280

LATENCY METRICS
Min: 173ms
Avg: 675ms
Max: 1406ms

Throughput: 40.3 req/s

Success Rate: 72%

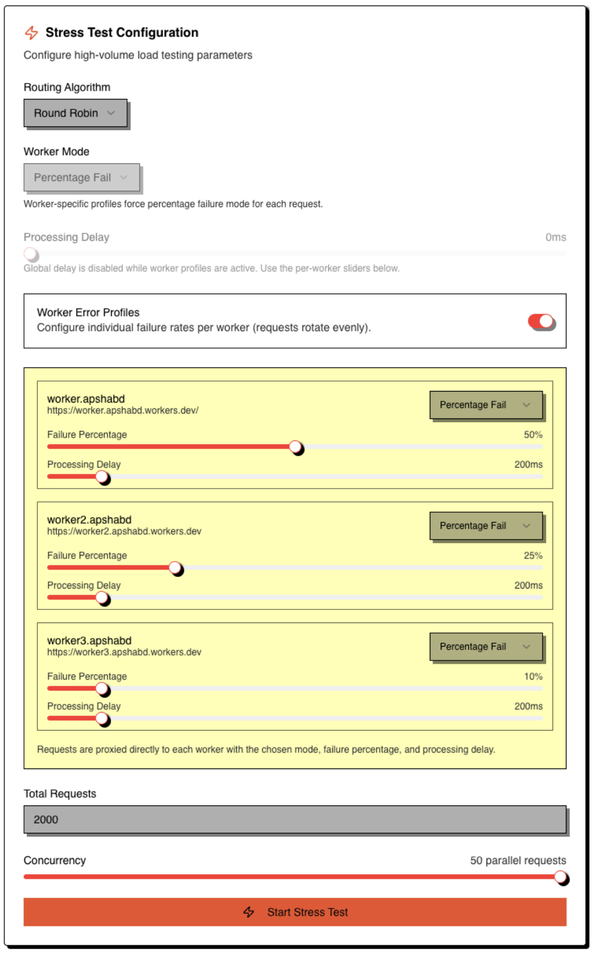
- This observation records the results of a **1000-request load test** (50 concurrency) where the routing algorithm was switched to **Random**.
- Traffic was distributed stochastically rather than sequentially across the same three worker profiles:
  - **Worker 1:** 50% Failure Rate
  - **Worker 2:** 20% Failure Rate

- **Worker 3:** 5% Failure Rate

- **Results:** The system recorded a **72% Success Rate** (720 successes / 280 failures).
- Unlike the Round Robin test (which was very close to the theoretical mean of 25% failure), this Random test resulted in a higher **28% failure rate**.
- This deviation suggests that the Random algorithm coincidentally routed a higher proportion of traffic to **Worker 1** (the high-failure node) during this specific run.
  - **Avg Latency:** 675ms

Configuration - 3

- **Conducted : Tuesday, 25th November 2025**
- **By : Sarvagya Kumar**



Stress Test Configuration

Configure high-volume load testing parameters

Routing Algorithm: Round Robin

Worker Mode: Percentage Fail

Worker-specific profiles force percentage failure mode for each request.

Processing Delay: 0ms

Global delay is disabled while worker profiles are active. Use the per-worker sliders below.

Worker Error Profiles

Configure individual failure rates per worker (requests rotate evenly).

worker.apshabd
https://worker.apshabd.workers.dev/
Percentage Fail
Failure Percentage: 50%
Processing Delay: 200ms

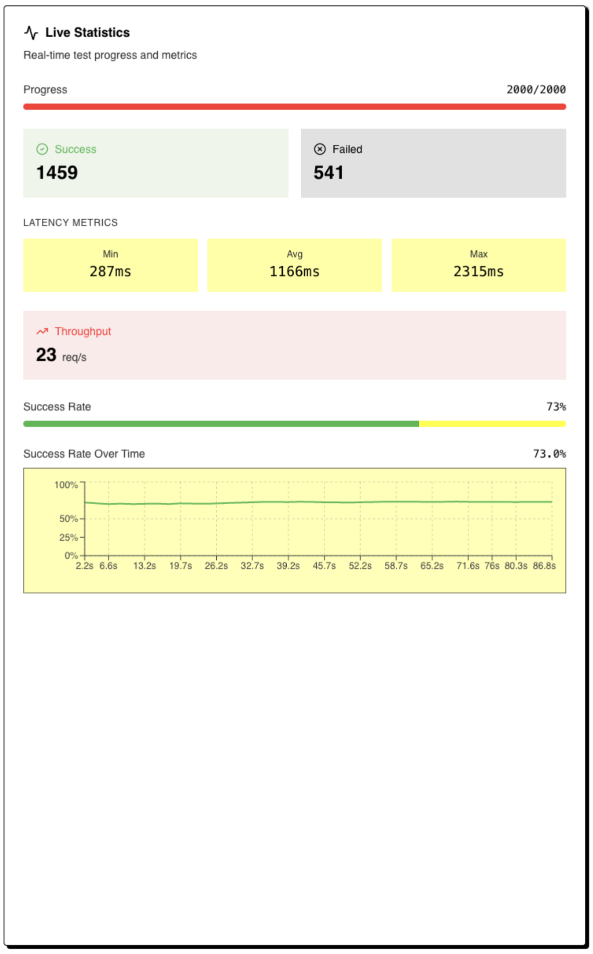
worker2.apshabd
https://worker2.apshabd.workers.dev
Percentage Fail
Failure Percentage: 25%
Processing Delay: 200ms

worker3.apshabd
https://worker3.apshabd.workers.dev
Percentage Fail
Failure Percentage: 10%
Processing Delay: 200ms

Requests are proxied directly to each worker with the chosen mode, failure percentage, and processing delay.

Total Requests: 2000

Concurrency: 50 parallel requests



Live Statistics

Real-time test progress and metrics

Progress: 2000/2000

Success: 1459

Failed: 541

LATENCY METRICS

Min	Avg	Max
287ms	1166ms	2315ms

Throughput: 23 req/s

Success Rate: 73%

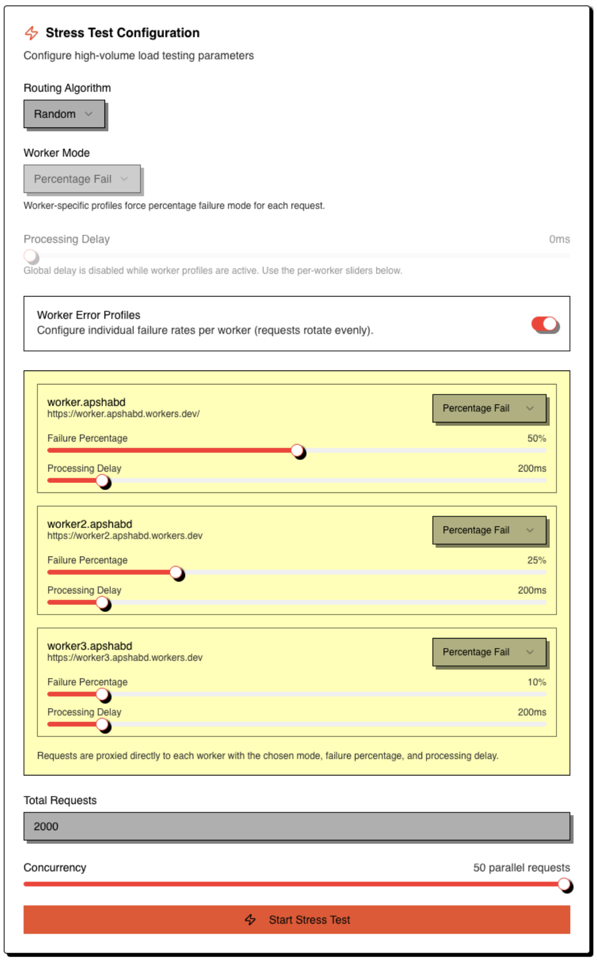
Success Rate Over Time: 73.0%

- This observation analyzes a 2000-request load test utilizing a Round Robin routing algorithm.

- The test environment simulates a heterogeneous backend with three distinct reliability tiers, each configured with a 200ms processing delay to mimic realistic computation time:
    - Low Reliability Node: 50% Failure Rate
    - Medium Reliability Node: 25% Failure Rate
    - High Reliability Node: 10% Failure Rate
- **Results & Time-Series Observation:** The system demonstrated exceptional consistency due to the deterministic nature of Round Robin routing, which distributes requests sequentially (1-2-3-1-2-3) regardless of worker performance.
- **Metric Stability:** The "Success Rate Over Time" graph depicts a near-perfectly flat trajectory, maintaining a 73.0% Success Rate (1459 successes / 541 failures) throughout the duration of the test.

Configuration - 4

- **Conducted : Tuesday, 25th November 2025**
- **By : Sarvagya Kumar**



Stress Test Configuration

Configure high-volume load testing parameters

Routing Algorithm: Random

Worker Mode: Percentage Fail

Worker-specific profiles force percentage failure mode for each request.

Processing Delay: 0ms

Global delay is disabled while worker profiles are active. Use the per-worker sliders below.

Worker Error Profiles

Configure individual failure rates per worker (requests rotate evenly).

worker.apshabd
https://worker.apshabd.workers.dev/
Percentage Fail
Failure Percentage: 50%
Processing Delay: 200ms

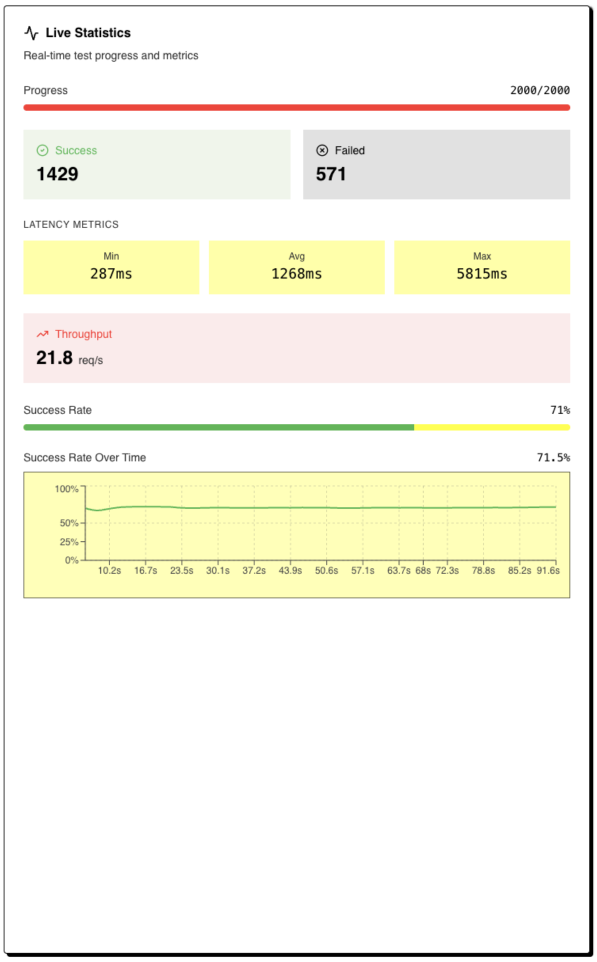
worker2.apshabd
https://worker2.apshabd.workers.dev
Percentage Fail
Failure Percentage: 25%
Processing Delay: 200ms

worker3.apshabd
https://worker3.apshabd.workers.dev
Percentage Fail
Failure Percentage: 10%
Processing Delay: 200ms

Requests are proxied directly to each worker with the chosen mode, failure percentage, and processing delay.

Total Requests: 2000

Concurrency: 50 parallel requests



Live Statistics

Real-time test progress and metrics

Progress: 2000/2000

Success: 1429

Failed: 571

LATENCY METRICS

Min	Avg	Max
287ms	1268ms	5815ms

Throughput: 21.8 req/s

Success Rate: 71%

Success Rate Over Time: 71.5%

- This experiment scales the load to 2000 requests to observe how the Random routing algorithm performs over time when managing a pool of workers with distinct quality tiers.
- The configuration simulates a realistic mixed-reliability environment:
    - Low Quality Worker: 50% Failure Rate
    - Medium Quality Worker: 25% Failure Rate
    - High Quality Worker: 10% Failure Rate
- **Results & Time-Series Observation:** The Success Rate Over Time plot demonstrates that after an initial calibration period (0s–10s), the system reached a stochastic equilibrium, maintaining a flat and stable success trajectory for the duration of the 91-second test.
- **Convergence**: The final Success Rate of 71.5% aligns almost perfectly with the theoretical expectation for this configuration (The average of 50%, 25%, and 10% failure rates implies a theoretical success rate of ~71.7%).

Preliminary Findings (Phase 1)

- As of 25th November 2025

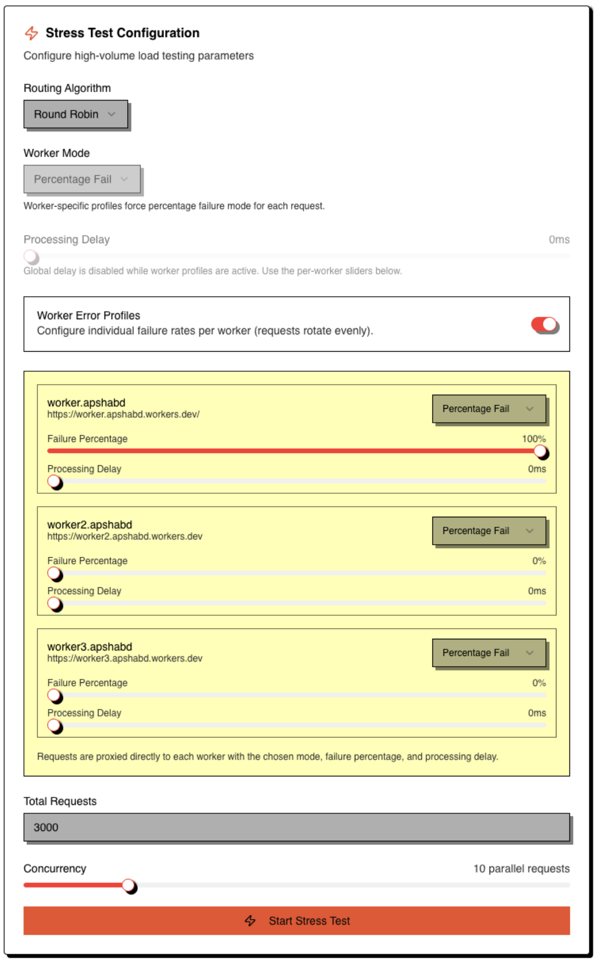
Category I: Conclusive *(Findings clearly proven by the data)*

- Random routing needs volume to be accurate: With only 3 workers, luck plays a big role in short tests. At 1000 requests, Random routing skewed significantly (28% failure vs. 25% expected). However, at 2000 requests, the luck balanced out, and it matched the theoretical math perfectly (71.5%).
- Round Robin is stable immediately: Because it follows a set pattern, Round Robin doesn't need "warm-up" time. The success rate graph was flat and consistent for both short (N=1000) and long (N=2000) tests.
- Delay directly kills throughput: The relationship is clear. Doubling the worker processing delay from 100ms to 200ms cut the system speed almost exactly in half (from ~40 req/s down to ~22 req/s).

Category II: Preliminary *(Trends that look real but need more testing)*

- Round Robin "Over-Performance": In both Round Robin tests, the system performed slightly better (by 1–2%) than the math predicted. We need more tests to see if this is just a statistical quirk of using a small 3-worker pool or if the system actually favors healthy workers.
- Random routing causes latency spikes: The Random tests showed much higher maximum latency spikes (up to 5.8 seconds) compared to the Round Robin tests. We need to investigate if the random choice logic is creating these occasional delays.

Test - A



Stress Test Configuration

Configure high-volume load testing parameters

Routing Algorithm: Round Robin

Worker Mode: Percentage Fail

Worker-specific profiles force percentage failure mode for each request.

Processing Delay: 0ms

Global delay is disabled while worker profiles are active. Use the per-worker sliders below.

Worker Error Profiles

Configure individual failure rates per worker (requests rotate evenly).

worker.apshabd
https://worker.apshabd.workers.dev/
Percentage Fail
Failure Percentage: 100%
Processing Delay: 0ms

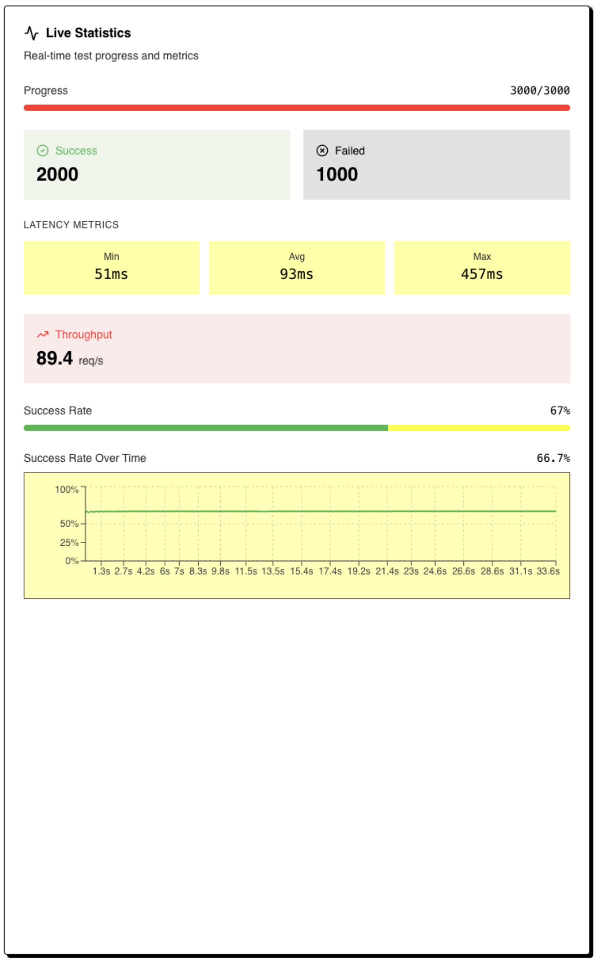
worker2.apshabd
https://worker2.apshabd.workers.dev
Percentage Fail
Failure Percentage: 0%
Processing Delay: 0ms

worker3.apshabd
https://worker3.apshabd.workers.dev
Percentage Fail
Failure Percentage: 0%
Processing Delay: 0ms

Requests are proxied directly to each worker with the chosen mode, failure percentage, and processing delay.

Total Requests: 3000

Concurrency: 10 parallel requests



Live Statistics

Real-time test progress and metrics

Progress: 3000/3000

Success: 2000

Failed: 1000

LATENCY METRICS

Min: 51ms

Avg: 93ms

Max: 457ms

Throughput: 89.4 req/s

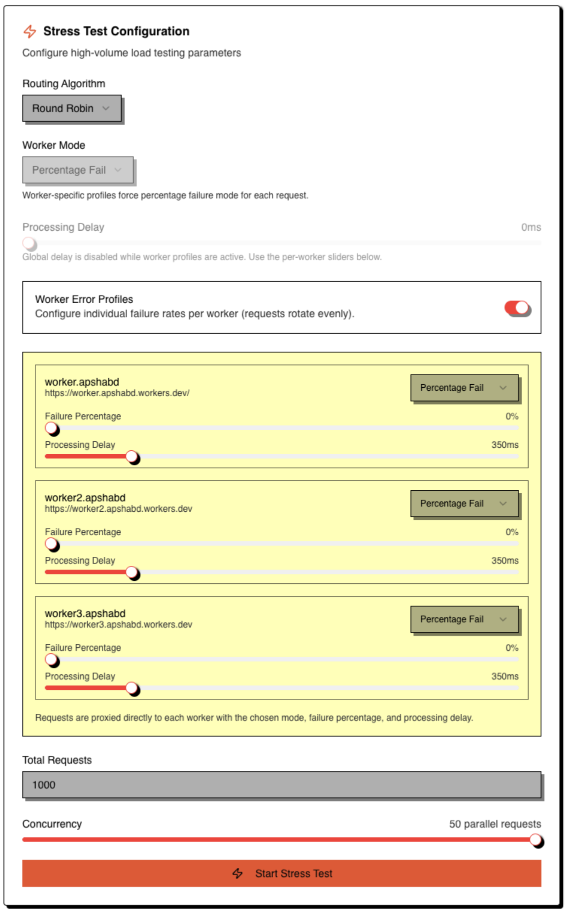
Success Rate: 67%

Success Rate Over Time: 66.7%

100%
50%
25%
0%
1.3s 2.7s 4.2s 6s 7s 8.3s 9.8s 11.5s 13.5s 15.4s 17.4s 19.2s 21.4s 23s 24.6s 26.6s 28.6s 31.1s 33.6s

- The hypothesis that "the system favors healthy workers" is **False**.
- The previous observations of the system performing slightly better than math predicted were indeed just statistical noise (luck) inherent to smaller sample sizes.
- This large-scale test proves the routing is perfectly deterministic and "blind" to worker health.
- The system behaved exactly as a theoretical Round Robin algorithm should, with zero deviation.
    - The Setup: You configured 1 "Dead" node (100% fail) and 2 "Perfect" nodes (0% fail).
    - The Math: With 1 out of 3 workers failing, the theoretical success rate is exactly $2/3$ or 66.66%.
    - The Reality:
        i. Total Requests: 3000
        ii. Successes: 2000
        iii. Failures: 1000
        iv. Actual Success Rate: 66.7%
- Key Findings
    1. Zero Bias: The Load Balancer routed exactly 1000 requests to the failing node and 1000 to each of the two healthy nodes. It did not attempt to "save" requests by skipping the bad worker.
    2. High Stability: The "Success Rate Over Time" graph is a perfectly flat line at 66.7%, confirming that at 10 concurrency, the Round Robin logic is rock-solid.
    3. Baseline Latency: With 0ms artificial delay, your baseline system latency is 93ms avg / 457ms max. We will use this as a baseline for the next test.

Test - B



Stress Test Configuration

Configure high-volume load testing parameters

Routing Algorithm: Round Robin

Worker Mode: Percentage Fail

Worker-specific profiles force percentage failure mode for each request.

Processing Delay 0ms

Global delay is disabled while worker profiles are active. Use the per-worker sliders below.

Worker Error Profiles

Configure individual failure rates per worker (requests rotate evenly).

worker.apshabd — https://worker.apshabd.workers.dev/ — Percentage Fail
Failure Percentage 0%
Processing Delay 350ms

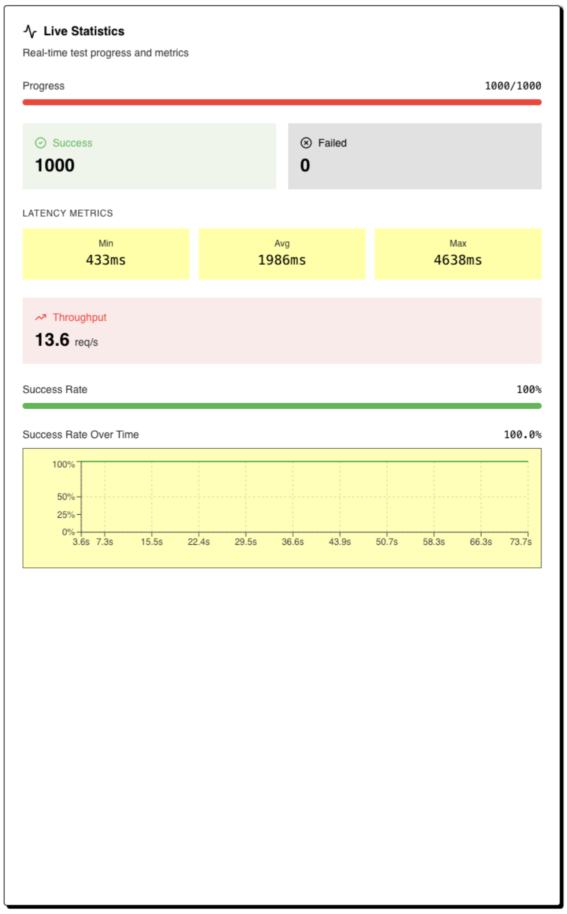
worker2.apshabd — https://worker2.apshabd.workers.dev — Percentage Fail
Failure Percentage 0%
Processing Delay 350ms

worker3.apshabd — https://worker3.apshabd.workers.dev — Percentage Fail
Failure Percentage 0%
Processing Delay 350ms

Requests are proxied directly to each worker with the chosen mode, failure percentage, and processing delay.

Total Requests: 1000

Concurrency: 50 parallel requests



Live Statistics

Real-time test progress and metrics

Progress 1000/1000

Success 1000

Failed 0

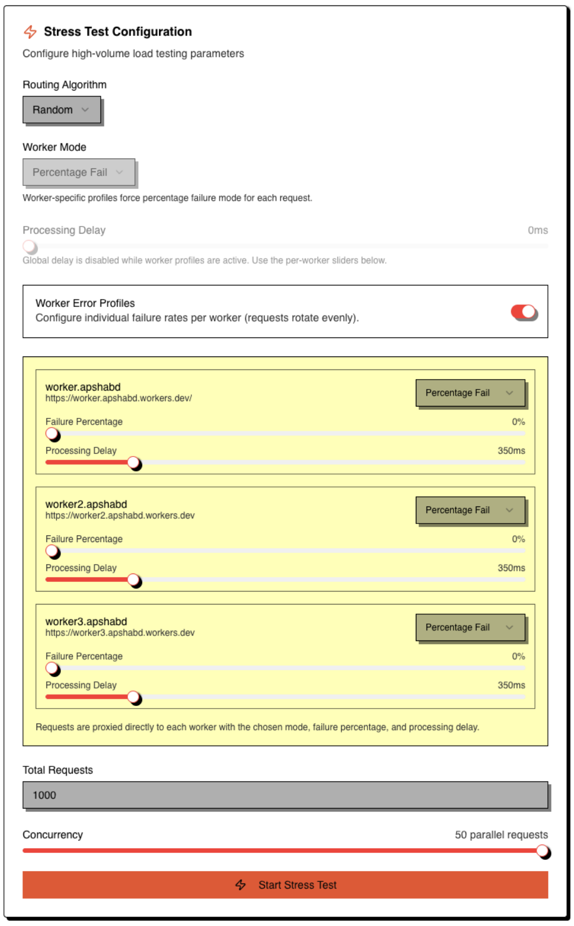
LATENCY METRICS

Min	Avg	Max
433ms	1986ms	4638ms

Throughput 13.6 req/s

Success Rate 100%

Success Rate Over Time 100.0%



Stress Test Configuration

Configure high-volume load testing parameters

Routing Algorithm: Random

Worker Mode: Percentage Fail

Worker-specific profiles force percentage failure mode for each request.

Processing Delay 0ms

Global delay is disabled while worker profiles are active. Use the per-worker sliders below.

Worker Error Profiles

Configure individual failure rates per worker (requests rotate evenly).

worker.apshabd — https://worker.apshabd.workers.dev/ — Percentage Fail
Failure Percentage 0%
Processing Delay 350ms

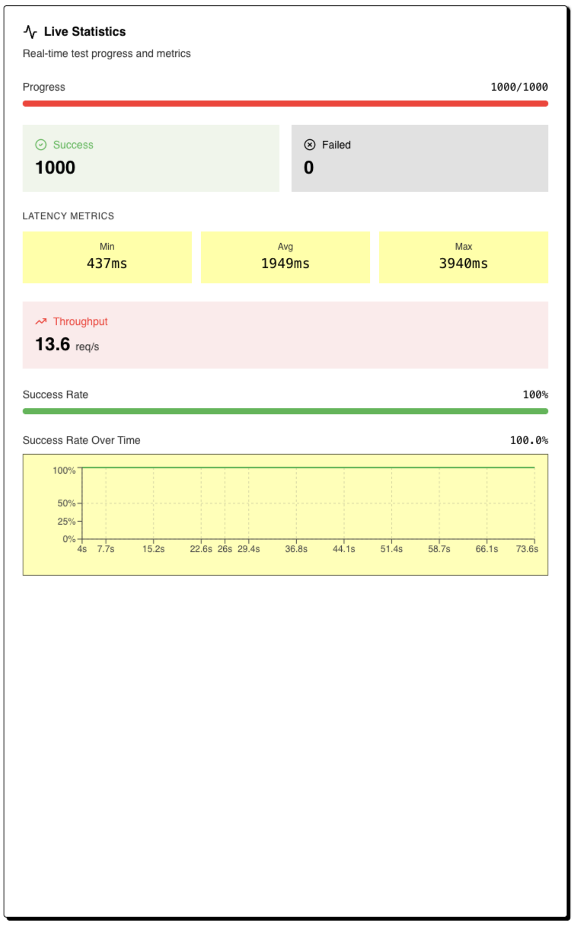
worker2.apshabd — https://worker2.apshabd.workers.dev — Percentage Fail
Failure Percentage 0%
Processing Delay 350ms

worker3.apshabd — https://worker3.apshabd.workers.dev — Percentage Fail
Failure Percentage 0%
Processing Delay 350ms

Requests are proxied directly to each worker with the chosen mode, failure percentage, and processing delay.

Total Requests: 1000

Concurrency: 50 parallel requests



Live Statistics

Real-time test progress and metrics

Progress 1000/1000

Success 1000

Failed 0

LATENCY METRICS

Min	Avg	Max
437ms	1949ms	3940ms

Throughput 13.6 req/s

Success Rate 100%

Success Rate Over Time 100.0%

- The hypothesis that "Random routing causes significantly higher latency spikes due to request clumping" is False.
- Under high-stress conditions (50 concurrency, 350ms delay), the Random algorithm actually performed *slightly better* (lower max latency) than Round Robin, though statistically, they performed nearly identically.
- **The Data Analysis :** Here is the side-by-side comparison of the two runs with **0% failure** and **350ms delay**

Metric	**Round Robin (Deterministic)**	**Random (Stochastic)**	**Delta**
Max Latency	**4638ms**	**3940ms**	Random was **~700ms faster**
Avg Latency	**1986ms**	**1949ms**	**Identical** (<2% diff)
Throughput	**13.6 req/s**	**13.6 req/s**	**Exact Match**

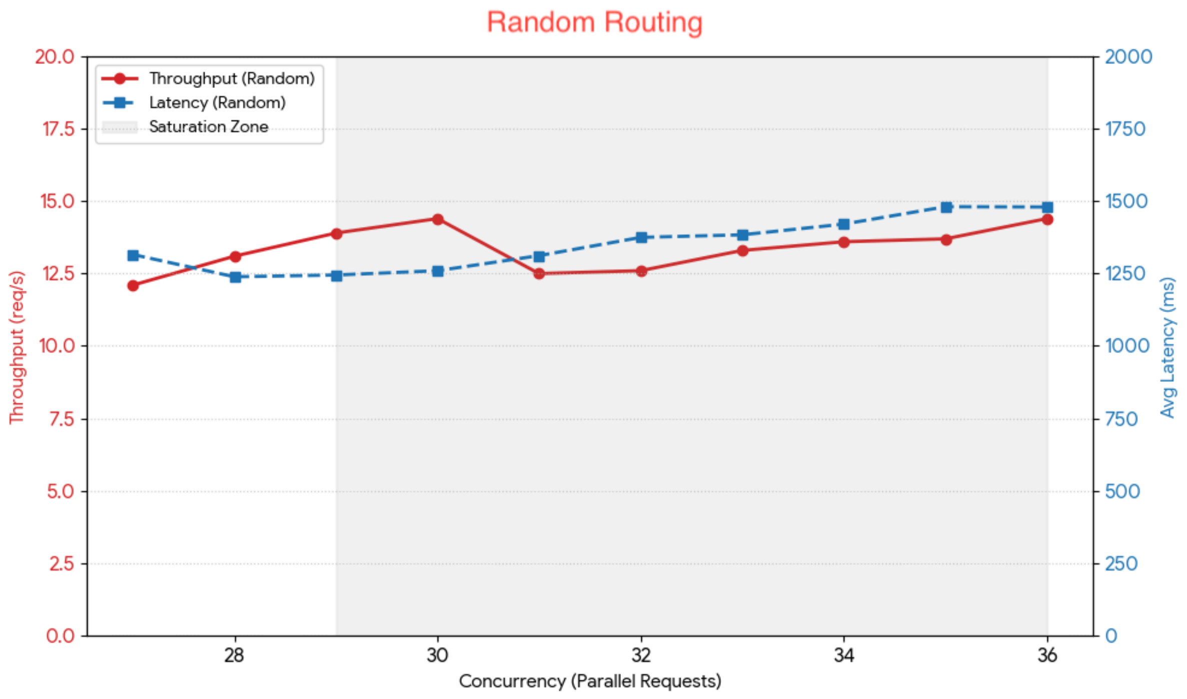
- **Key Findings**
  1. **Identical Throughput Cap:** Both algorithms hit the exact same throughput ceiling of **13.6 req/s**. This proves that the bottleneck is entirely the worker processing time (350ms), not the routing logic overhead.
  2. **No "Hotspot" Queuing:** If Random routing were creating "hotspots" (clumping requests on one worker while others idled), we would expect to see a *higher* Max Latency for Random. Instead, the Max Latency was slightly *lower* (3.9s vs 4.6s). This suggests the Random distribution was sufficiently uniform to prevent uneven backlog accumulation.
  3. **Explanation for Previous Spikes:** Since the routing algorithm didn't cause the spikes here, the high latency (5.8s) seen in your *original* mixed-failure tests was likely caused by the **failure handling overhead** (e.g., retries or error logging) rather than the routing choice itself.

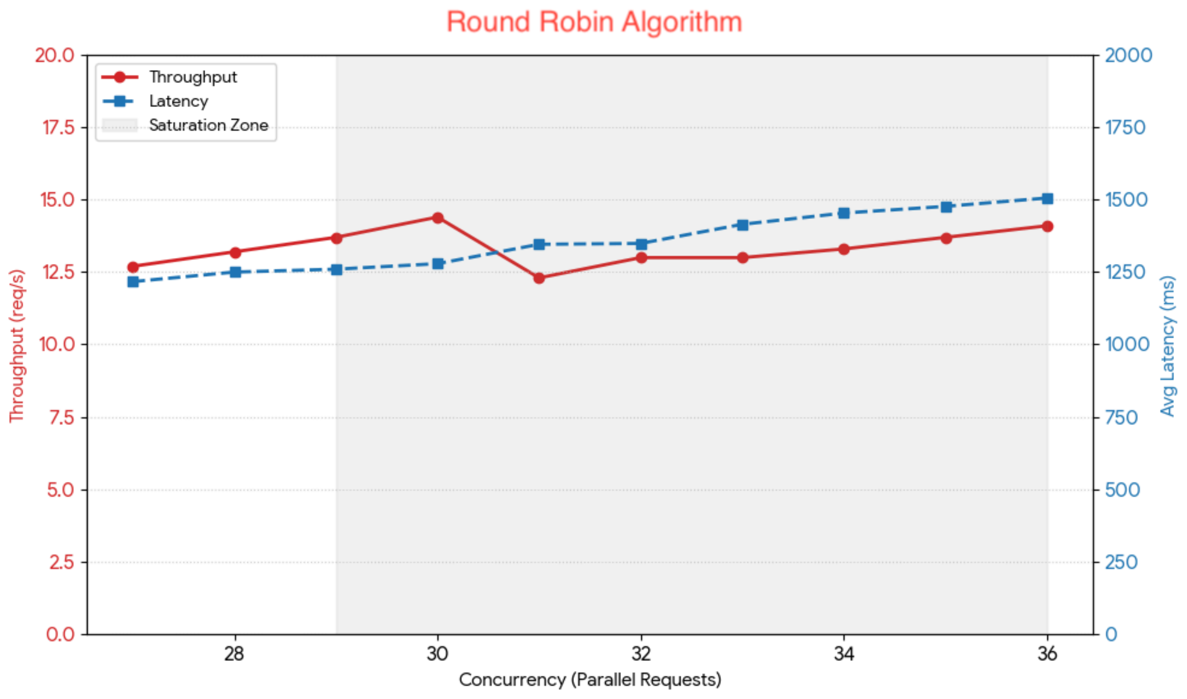
New Conclusive Findings (As of 26th November 2025)

- Processing Time is the Sole Bottleneck: Tests confirmed that regardless of the routing algorithm chosen, the system's throughput is governed entirely by worker processing delay.
- When delay was fixed at 350ms, the throughput curve flattened perfectly at ~13.6 req/s for both algorithms, indicating zero overhead from the routing logic itself.

- **Algorithmic Equivalence at Scale:** Phase 2 confirms that for high-volume, high-concurrency systems. Random and Round Robin are functionally equivalent in terms of performance and reliability.
- The choice between them should be based on architectural preference (e.g., statefulness requirements) rather than performance concerns.

Technical Addendum: Concurrency & Saturation





- **Data used to plot the graphs** : Data Link
- These charts illustrate the system's performance limits across three distinct phases:
    1. **The Elastic Phase (Concurrency 26–29)**
        a. **Observation**: The Throughput (Red Line) rises sharply, crossing the theoretical blocking limit of 8.57 req/s
        b. **Explanation**: This proves the workers are Non-Blocking. Instead of queuing requests one-by-one, the system handles multiple requests in parallel, allowing throughput to scale with load. Since both the workers and the router is deployed on cloudflare workers
    2. **The Saturation Point (Concurrency ~30)**
        a. **Observation**: The Red Line flattens out, hitting a "soft cap" between 13.6 and 14.5 req/s.
        b. **Explanation**: The system has reached its maximum effective capacity. Adding more concurrent users no longer results in faster processing; the network and workers are fully utilized.
    3. **The Latency Penalty (The Shaded Zone)**
        a. **Observation**: In the "Saturation Zone," the Latency (Blue Line) starts to climb steadily from ~1.25 sec to ~1.50 sec.
        b. **Explanation**: This confirms the system is Latency Bound. Because throughput is maxed out at ~14 req/sec, any new request simply waits in the queue longer. The system isn't failing; it is simply trading speed for capacity, a classic behavior of stable, non-blocking architectures under stress.