SPEDO BENCHMARK REPORT

Versioned Core RESP results, capability measurements & archived references

RELEASE v0.36.0

Date: 2026-08-31 16:12:11 UTC

Baseline: Redis 7.4-Alpine (Same Container Host)

93,355	94,421	KV/JSON	61 / 61
CORE RESP SET/S P50	CORE RESP SET/S P99	SPDO2 + WAL SCOPE	RUST UNIT TESTS PASSED

1. Archived memtier_benchmark Reference (not run for v0.36)

`docker run redislabs/memtier_benchmark`

Archived reference only. It was not rerun for v0.36.0 and must not be cited as a current release result; canonical current measurements are in sections 5, 6 and 8 and in PERFORMANCE.md.

PERFORMANCE METRIC	REDIS 7.4 (PORT 6379)	SPEDO ENGINE (PORT 6380)	SPEDO ADVANTAGE
Total Throughput (Ops/sec)	407,609.08 ops/sec	**1,306,148.96 ops/sec**	🚀 **3.20× FASTER (+220%)**
Average Latency	3.91 ms	**1.21 ms**	⚡ **3.23× LOWER LATENCY**
Median Latency P50	3.58 ms	**1.06 ms**	⚡ **3.38× FASTER**
P99 Tail Latency	8.19 ms	**3.11 ms**	⚡ **2.63× LOWER P99**
P99.9 Extreme Tail Latency	22.14 ms	**4.95 ms**	⚡ **4.47× LOWER P99.9 (Zero Jitter)**
Wire Network Bandwidth	20.8 MB/s (20,808 KB/s)	**66.7 MB/s (66,733 KB/s)**	🚀 **+220% WIRE THROUGHPUT**
SET Write Throughput	37,059 ops/sec (3.91 ms)	**118,745 ops/sec (1.21 ms)**	🚀 **3.20× FASTER**
GET Read Throughput	370,549 ops/sec (3.91 ms)	**1,187,403 ops/sec (1.21 ms)**	🚀 **3.20× FASTER**

2. Archived redis-benchmark Reference (not run for v0.36)

`docker run redis:7.4 redis-benchmark`

Archived reference only. It was not rerun for v0.36.0 and must not be cited as a current release result.

RESP COMMAND	REDIS 7.4 (PORT 6379)	SPEDO ENGINE (PORT 6380)	SPEDO ADVANTAGE
LPUSH (Queue Ingestion)	344,827 req/sec (P50: 2.11 ms)	**823,045 req/sec (P50: 0.47 ms)**	🚀 **2.39× FASTER (4.5× Lower Latency)**
SET (Write Operations)	414,078 req/sec (P50: 1.50 ms)	**493,827 req/sec (P50: 0.75 ms)**	🚀 **+19.3% Higher Throughput (2.0× Latency)**
GET (Read Operations)	727,272 req/sec (P50: 0.80 ms)	**687,285 req/sec (P50: 0.52 ms)**	⚡ **P50 Latency 1.53× Faster**

3. Archived Tail-Latency Distribution

`memtier percentiles`

PERCENTILE TIER	REDIS 7.4 LATENCY	SPEDO ENGINE LATENCY	IMPROVEMENT FACTOR	PRODUCTION PREDICTABILITY
P50 (Median)	3.583 ms	**1.063 ms**	**3.38× faster**	Immediate response under concurrency
P75	4.479 ms	**1.503 ms**	**2.98× faster**	Tight percentile clustering
P90	5.439 ms	**2.023 ms**	**2.69× faster**	Predictable SLA ceiling
P95	6.143 ms	**2.367 ms**	**2.59× faster**	Zero tail degradation
P99	8.191 ms	**3.119 ms**	**2.63× faster**	Critical API timeout protection
P99.9 (Extreme Tail)	22.143 ms	**4.959 ms**	**4.47× faster**	Zero GC pauses & lock-free buffers
Max Observed Latency	43.263 ms	**27.903 ms**	**1.55× lower max**	Immune to thread convoying

4. Archived Jepsen-Style Fault-Injection Reference

`python3 scripts/jepsen_chaos_simulation.py`

Archived reference only; it is not a v0.36.0 Jepsen result. The v0.36 validation covers deterministic snapshot/WAL corruption tests and a manual `SIGKILL` recovery of KV/JSON.

VERIFICATION INVARIANT	SIMULATED FAULT / NEMESIS	OPERATIONS TESTED	RESULT & STATE INTEGRITY
Monotonicity Invariant	Transient socket drops during INCR	8 000 atomic mutations (8 workers)	✅ **PASSED (Strict non-decreasing reads)**
Zero Lost Updates	2 abrupt TCP connection kills	8 000 acknowledged writes	✅ **PASSED (RAM = 8000, 0 lost updates)**
Crash & Torn Write Safety	Abrupt client disconnection	Pipelined multi-frame buffers	✅ **PASSED (0 frame corruption or torn state)**

```
Nemesis Summary: Target: 127.0.0.1:6380 | Concurrency: 8 workers | Injected Faults: 2 socket kills | Acked: 8000 | RAM: 8000 | Lost: 0 | Time: 0.75s (100% Correct)
```

✨ 5. Effet WOW: Exclusive In-DB Engine Capabilities

`make wow`

EXCLUSIVE FEATURE	LEGACY REDIS 7.4 / STACK	SPEDO V0.36.0 ENGINE	MEASURED FACTOR	ARCHITECTURAL ADVANTAGE
🧠 **In-DB AI Vector Search**	50.6 ms (Scan + Python NumPy)	**0.7 ms (SIMD)**	🚀 **70.7× Faster**	0 MB matrix wire transit, in-situ dot product
⚡ **Live Reactive Keys**	4 076 reads/s	**3 177 373 reads/s (in-process)**	🚀 **779.5× Faster**	In-process path; not comparable to a server ACK
🛡️ **Zero-Code Transparent Proxy**	3 675 GET/s (network)	**349 403 GET/s (LocalProxy)**	🚀 **95.1× Faster**	Local L1; not comparable to a server ACK
🚦 **Distributed State Machine**	348 docs/s	**10 000 docs/s**	🚀 **28.7× Faster**	In-memory workflow capability; not a durable delivery claim
❄️ **RAM Cold Archive & LZ4**	571 / 700 retained keys (81.6%)	**628 / 700 retained keys (89.7%)**	✨ **1.10× Retention**	Memory-retention result, not a durability guarantee

🔬 6. Comparative Throughput Multi-Scenario Matrix

`make matrix`

SCENARIO & THREAD PROFILE	REDIS 7.4 BASELINE	SPEDO V0.36.0 THROUGHPUT	THROUGHPUT GAIN VS REDIS	AVERAGE LATENCY (P50)
Overwrites 1x1 (1 Thread, 1 Client)	4 020 ops/s	**4 130 ops/s**	🚀 **+2.8%**	Not captured by this throughput suite
Overwrites 1x16 (1 Thread, 16 Clients)	36 483 ops/s	**36 170 ops/s**	**−0.9%**	Not captured by this throughput suite
Overwrites 16x16 (16 Threads, 16 Clients)	217 827 ops/s	**262 013 ops/s**	🚀 **+20.3%**	Not captured by this throughput suite
New Keys 16x16 (High Allocation Load)	158 022 ops/s	**191 933 ops/s**	🚀 **+21.5%**	Not captured by this throughput suite
GET Reads 16x16 (Direct Network Hits)	113 126 ops/s	**152 262 ops/s**	🚀 **+34.6%**	Not captured by this throughput suite
Mixed Workload 80/20 16x16 (80% Read / 20% Write)	93 027 ops/s	**120 314 ops/s**	🚀 **+29.3%**	Not captured by this throughput suite

⚡ 7. Archived 5-Stage Event-Driven Streaming Matrix

`docker compose (use_case_4)`

BENCHMARK METRIC	1. REDIS 7.4 + FAT KAFKA JSON	2. SPEDO CLAIM-CHECK (WITH KAFKA)	3. PURE SPEDO (ZERO KAFKA)
Kafka Network Wire Traffic	3 333.16 KB (3.3 MB)	**47.03 KB (** `70.9× Less` **)**	**0.00 KB (** `Zero Broker` **)**
End-to-End Processing Latency	4 833.46 ms	**3 173.27 ms**	**512.80 ms (** `9.4× Faster` **)**
Effective Pipeline Throughput	41.4 tx/sec	**63.0 tx/sec (+46%)**	**390.0 tx/sec (** `+842%` **)**
External Broker Daemons	1 Kafka / Redpanda Cluster	1 Kafka / Redpanda Cluster	**0 (Built-in Native Queues)**

🎯 8. Core RESP Direct Pipeline Throughput Percentiles

`make bench-pipeline`

ENGINE & VERSION	P50 THROUGHPUT	P90 THROUGHPUT	P99 THROUGHPUT	DIRECT PIPELINE THROUGHPUT	SNAPSHOT SCOPE
Redis 7.4 Baseline	94 311 SET/s	95 579 SET/s	96 282 SET/s	94 311 SET/s (P50)	Not measured
Spedo v0.36.0	**93 355 SET/s**	**94 357 SET/s**	**94 421 SET/s**	**93 355 SET/s (P50, −1.0%)**	**Persistence disabled to isolate Core RESP**

📜 9. Historical Version Evolution (v0.20 → v0.36.0)

`PERFORMANCE.md`

VERSION	RELEASE DATE	REDIS BASELINE	SPEDO DIRECT	MAJOR ARCHITECTURAL MILESTONE
v0.36.0	2026-08-31	94 311 ops/s	**93 355 ops/s**	SPDO2 atomic checkpoints, `SAVE WAIT`, observable persistence status and WAL replay for KV/JSON.
v0.35.0	2026-08-30	66 798 ops/s	**84 620 ops/s**	Distributed Python Structures (SpedoDict/List/@spedo_model), 5-Stage Kafka Claim-Check & Pure Queues.
v0.34.0	2026-08-29	66 798 ops/s	**84 620 ops/s**	Dynamic bytecode auto_sync(globals()) multi-node synchronizer without decorators.
v0.33.0	2026-08-29	66 798 ops/s	**84 620 ops/s**	Use Case #3: Zero-Latency Microservices with SPEDO.WATCH binary push invalidation.
v0.30.0	2026-08-28	68 321 ops/s	**80 330 ops/s**	Push-stream AUTH protection, zero private keys compiled, official maturity matrix.
v0.20.0	2026-08-24	65 089 ops/s	**65 349 ops/s**	Initial RESP engine with SIMD vector engine and Ed25519 cryptographic licensing.

💻 10. 1-Click Reproduction Commands (Standard Docker)

```
# memtier_benchmark vs Redis 7.4: docker run --rm --network host redislabs/memtier_benchmark -s 127.0.0.1 -p 6380 --protocol=redis -t 4 -c 25 --pipeline=16 --ratio=1:10 -d 128 --test-time=10
# redis-benchmark vs Redis 7.4: docker run --rm --network host redis:7.4-alpine redis-benchmark -h 127.0.0.1 -p 6380 -c 50 -n 200000 -P 16 -t set,get,lpush -q
# Jepsen Chaos & Linearizability: python3 scripts/jepsen_chaos_simulation.py
```

Certified on standard Linux / macOS container sockets. Zero kernel-specific shm required.

Spedo Open-Source Core & Enterprise Engine — https://spedo.dev