Hakozuna: A Lock-Free Box-Oriented Memory Allocator

Author: Tomoaki Kurata (TOMOAKI KURATA)
Contact: charmpic moecharm.dev@gmail.com
X (Twitter): https://x.com/CharmNexusCore
Version: 2026-02-18 v3.0

Abstract

We propose Hakozuna (ACE-Alloc, hz3), a small-object allocator competitive with mimalloc and tcmalloc, and hz4, a remote-heavy message-passing profile developed after the hz3 line. In our evaluation on Ubuntu (native), conducting SSOT of random_mixed, hz3 was the fastest (359.6M ops/s) and achieved the lowest RSS under R=0 local-only conditions. Under remote-heavy conditions, hz4 (message-passing version) showed superiority, demonstrating that the winning strategy divides according to usage. The core of the design is boundary concentration based on Box Theory, separating the hot path (TLS) from the control layer (refill/drain) and limiting transition points to a single location. Free routing defaults to **PTAG32**, and while the PTAG method is switchable at build time, this evaluation was conducted with **PTAG32 as default**. We validated the design with over 20 A/B experiments (including NO-GO cases) and ensured optimizations are reversible via compile-time flags. This evaluation fixed the ACE/ELO/CAP learning layers to FROZEN (default OFF) to measure the performance of the hz3 core. This study is further positioned as a demonstration case where **collaboration between a single developer and LLMs achieved performance exceeding state-of-the-art allocators in a short period of about 3 months.**

1. Introduction

Small-object allocation directly affects throughput and tail latency in real applications. Conventionally, this often remained a trade-off between "speed" and "memory efficiency," but this study stacks performance while maintaining observability and reversibility through design partitioning based on Box Theory. In designing, we used mimalloc and tcmalloc as primary references and comparison targets, verifying performance and stability in evaluations. A feature of this study is reaching SOTA-class performance in a short period through high-speed iteration of design, implementation, and evaluation with LLM collaboration. In the later phase, we intentionally moved from a single-profile goal to a dual-profile strategy: hz3 for local-heavy workloads and hz4 for remote/high-thread workloads.

1.0 Development Timeline (Summary)

- **2025 - 10 - 22**: Initial benchmark recorded (mid 2–32KB comparison).
- **2025 - 10 - 26**: Discovered a critical measurement error (measuring glibc malloc) and fixed the benchmark infrastructure. The true performance restarted from a point **8.8× slower** than mimalloc.
- **2026 - 01 - 24**: Reached a level where hz3/hz4 competes with tcmalloc/mimalloc in SSOT measurement. In R=0, hz3 is fastest, and in R=90, hz4 competes, clarifying the winning strategies by usage.

The contributions of this paper are:

- Boundary concentration (single transition point) based on Box Theory and "reversible" A/B design
- Low-overhead design centered on Two-Speed Tiny Front
- Dual-profile strategy: `hz3` (local/redis-oriented) and `hz4` (remote/high-thread-oriented)
- Multi-faceted evaluation including real applications and MT remote

1.1 Related Work

We compared against representative existing allocators: tcmalloc (classic: gperftools [1], modern: google/tcmalloc [4]), jemalloc [2], and mimalloc [3]. tcmalloc centers on thread cache and central freelist, performing load transfer between threads via transfer cache. jemalloc manages memory with tcache and decay/purge based on arena/extent. mimalloc adopts per-thread heap and delayed free, suppressing contention with segment management and abandoned processing. hz3 (ACE-Alloc) learns from these designs but differentiates itself by combining boundary concentration via Box Theory, **switchable PTAG routing**, separation of remote-free via RemoteStash/OwnerStash, and observability via A/B switching and SSOT.

2. Design Overview

2.1 Applying Box Theory

In this study, places where "role, responsibility, and ownership" intersect are separated as boxes, aggregating transition points to a single location. A box is a "human boundary," confining ownership and contention control inside the box. For example, the Remote Queue is limited to push only, and the Owner Stash handles reclamation only. Only at the adoption boundary (refill path) do drain → bind → owner occur collectively, clarifying the scope of impact and rollback.

2.2 Two-Speed Tiny Front

- HOT: Centered on TLS cache, processing with minimal instructions and branches
- WARM: Absorbs requests via batch replenishment (refill)
- COLD: Handles Superslab / Registry / OS cooperation

Learning and observation layers are isolated in upper boxes and do not affect the HOT path.

3. Implementation

3.1 Design Principles

hz3 enforces the following 4 principles based on Box Theory:

1. **Boundary Concentration**: Minimize boundaries between hot path and control layer, aggregating transition points to one location (refill path).
2. **Reversibility**: Design that allows optimizations to be reverted via A/B switching with compile-time flags.
3. **Observability**: Ensure reproducibility via one-shot logs (SSOT) at atexit.
4. **Fail-Fast**: Detect invalid states at boundaries and terminate abnormally early.

3.2 Lane Separation (fast / scale)

hz3 provides two build configurations depending on usage:

- **fast lane**: 8 shards, dense bank structure (TLS ~48KB), emphasizes low latency.
- **scale lane**: 32 shards, sparse ring structure (TLS ~6KB), supports high parallelism (S41 implementation record).

3.3 Three-Layer Structure

(1) Front Layer (Fast Path) **Role**: Allocation and freeing from TLS cache with minimal instructions and branches.

- **Alloc Fast Path**: size → size class conversion → pop from TLS local bin
- **Free Fast Path**: routing allows switching of **PTAG method**
  - PTAG32: pointer → page index conversion (1 arithmetic operation)
  - obtain (bin, dst) with tag read (1 memory access)
  - Default is PTAG32 (scale lane)

Optimization Technologies: - **S122 Split Count**: bin count update divided by 16, reducing RMW. - **S40 SoA Layout**: separation of head array and count array, improving cache line efficiency.

(2) Cache Layer (Cache Management) **Role**: Batch transfer between front and central layers, reducing lock contention.

- **TCache (Thread Cache)**: Holds Hz3TCache dedicated to each thread in TLS, caching max 16 per size class.
- **Owner Stash (S44)**: Intermediate buffer during remote free (MPSC: Multiple Producer Single Consumer)
  - Add remote free from other threads via CAS (lock-free)
  - Bulk acquire from Owner Stash before central access during own thread's alloc miss
  - Effect: Avoids mutex contention, improves throughput under remote-heavy conditions
- **RemoteStash Ring (S41, scale lane)**: TLS reduction via sparse structure (approx. 97% reduction vs dense 32-shard, S41 implementation record)

(3) Central Layer (Shared State) **Role**: Manages object pools shared between threads.

- **Central Bin**: Selectable lock-free (S142) or mutex-protected freelist. scale lane default is lock-free, bulk acquire/push via atomic exchange.
- **Small v2 (self-describing)**: Self-describing design requiring no segmap
  - SegHdr: Place magic/owner/page_meta at the head of each 2MB segment
  - PageHdr: Place magic/sc/owner at the head of each 4KB page
- **PageTagMap (PTAG32)**: Fast reverse lookup of size class and owner from pointer
  - Provides bin/dst with 32bit tag (bin: 16bit, dst: 8bit) or flat 32bit
  - Update: atomic store during page allocation

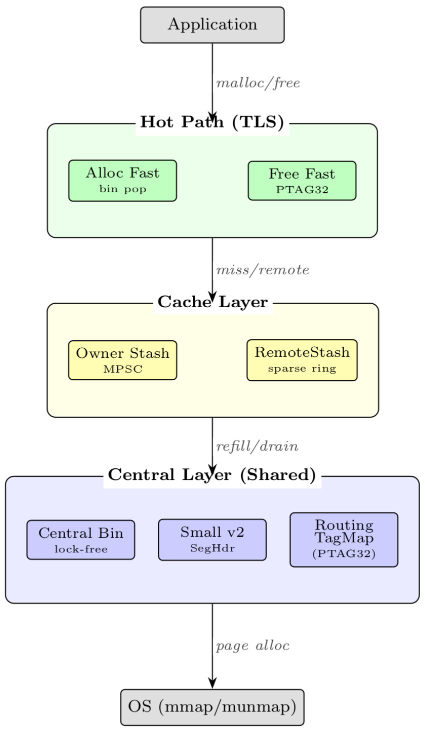

Figure 1: Figure 1: hz3 Overall Architecture

3.3.1 Summary of Boundaries (Transition Points)

Transition points between layers are aggregated in the refill path, prohibiting side effects across boundaries.

- **Hot → Cache**: miss / remote only. Does not touch shared state.
- **Cache → Central**: refill / drain only. Publishing and owner changes are aggregated here.
- **Central → OS**: Only acquisition/return of segment / page. Does not bring state into the Hot side.

3.4 Remote Free Processing

Remote free goes through the following stages:

1. **Detection**: Compare dst obtained from PTAG32 with my_shard.
2. **Temporary Storage**: Push to RemoteStash or outbox (thread-local operation).
3. **Flush**: When budget exceeded or at epoch
    - S41: Retrieve (dst, bin, ptr) from sparse ring
    - S44: CAS push to owner_stash (lock-free)
    - fallback: Push to central bin via mutex

4. **Reclamation**: Bulk pop from owner_stash via atomic exchange during owner thread's alloc miss.

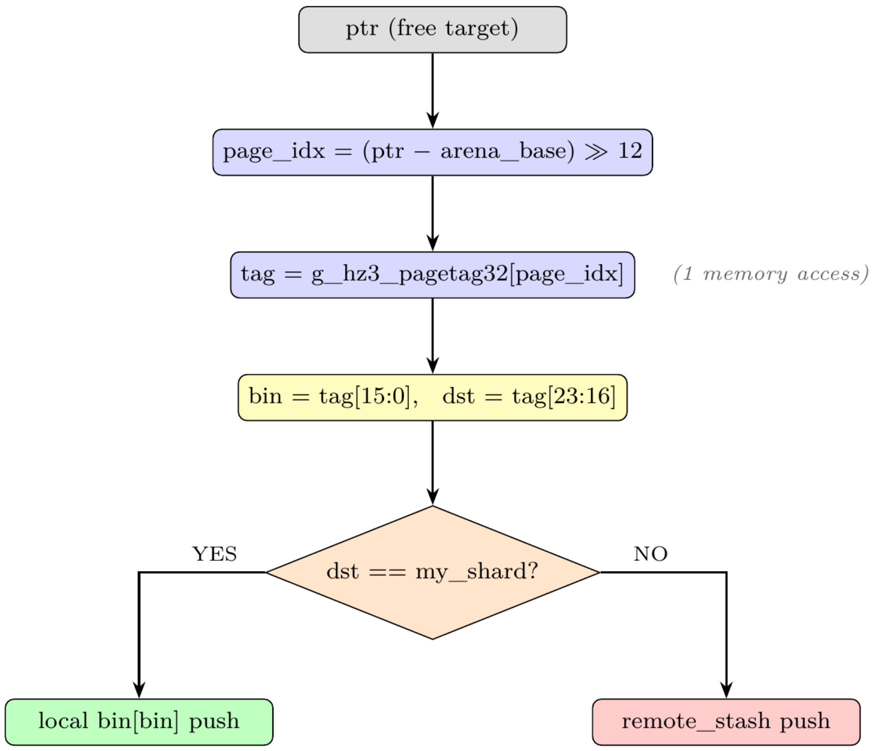
ptr (free target)
page_idx = (ptr − arena_base) ≫ 12
tag = g_hz3_pagetag32[page_idx]
(1 memory access)
bin = tag[15:0], dst = tag[23:16]
dst == my_shard?
YES
NO
local bin[bin] push
remote_stash push

Figure 2: Figure 2: PTAG32 Free Fast Path

Major Optimizations: - **S112 Full Drain Exchange**: Removed bounded drain retry+re-walk with atomic_exchange (improved xmalloc-test. See `docs/HAKMEM_HZ3_PAPER_NOTES.md` for details). - **S41 Sparse RemoteStash**: TLS approx. 97% reduction, maintaining ST performance. - **S44 Owner Stash**: Reduced mutex contention under remote-heavy conditions.

NO-GO Cases (Optimizations not adopted): - **S121 Series Page-Local Remote**: Regression in real workloads due to dependence on synthetic (xmalloc-test). - **S143 PGO**: Regression in r90 due to profile mismatch.

3.5 Positioning of Learning Layer (ACE / ELO)

The learning layer is confined in a "policy box" and does not affect the Hot Path.

- **FROZEN is default** (Learning OFF)
- **Hot Path refers only to "current policy values"**
- Learning ON/OFF is switchable via ENV (reversible design)

PNGs are embedded in the text, and TikZ manuscripts are saved in `docs/paper/ACE-Alloc/figures/`.

4. Evaluation

Detailed evaluation logs are aggregated in `docs/benchmarks/2026-02-18_PAPER_BENCH_RESULTS.md`. This paper presents only key results.

4.0 Evaluation Protocol

- **Fixed Environment**: CPU/OS/compiler/flags use those recorded in `docs/benchmarks/2026-02-18_PAPER_BENCH_RESULTS.md`.
- **Comparison under Identical Conditions**: Benchmarks are executed with the same binary and parameters, swapping the allocator via LD_PRELOAD.
- **Statistical Processing**: RUNS=10 is standard, adopting the median (long-running tests use RUNS=5).
- **Log Consistency**: Record git commit / build flags / execution parameters in SSOT header.
- **RSS**: Record max RSS with getrusage(2) ru_maxrss.
- **Evaluation Tables (LaTeX)**: Summarized in `docs/paper/ACE-Alloc/tables.tex`.

4.1 SSOT (random_mixed, T=16/R=90, RUNS=10)

See Table 2. In SSOT, hz4 was the fastest, with hz3/tcmalloc/mimalloc closely following.

4.2 T sweep (R=90, RUNS=3)

See Table 3. At T=16, hz4 extends its lead significantly, confirming the scaling effect of the message-passing design.

4.3 R sweep (T=16, RUNS=3)

See Table 4. At R=0, hz3 is fastest; at R=50, tcmalloc is fastest; at R=90, hz4 is fastest, showing that winning strategies divide according to the remote ratio.

4.2.1 Additional Verification Changing 64KB to Mid Treatment (OOM Countermeasure)

By moving 64KB from Large (mmap) to Mid, OOM under full-range conditions was resolved, confirming improvements of **+10.3%** at T=8/R=90, **+4.6%** at T=16/R=90, and **+4.2%** at T=16/R=0. Details are summarized in `docs/benchmarks/2026-01-23_hz3_64kb_mid_fix.md`.

4.4 ST dist_app (RUNS=3)

See Table 5. In ST, tcmalloc is fastest, followed closely by hz3.

4.5 RSS (ru_maxrss)

See Table 6 and Table 7. hz3 showed the minimum RSS in both MT remote and ST dist_app.

4.6 Memory-Release Tuning (Appendix)

Table 8 shows the SSOT comparison (T=16/R=90, RUNS=10) under memory-saving configurations. `mimalloc` with purge=0 achieved the lowest RSS (0.52GB) while sustaining 112M ops/s. By contrast, `hz3` with S64 ON was slightly worse than baseline in both throughput and RSS, indicating that S64 is a trade-off rather than a universal win. Light tuning (A: purge delay=8) improves RSS over default S64 while keeping ~104M ops/s (see `docs/benchmarks/2026-01-24_S64_LIGHT_TUNING_RESULTS.md`).

4.7 Larson sanity (S62 OFF)

In Larson hygienic, hz3 recorded 143.1M ops/s, confirming it operates at a level equivalent to tcmalloc (see results log for details).

4.8 mimalloc-bench subset (RUNS=5, Appendix)

See Table 9. hz3 was fastest in xmalloc-testN / sh8benchN / cache-scratchN, while hz4 was fastest in alloc-testN.

4.9 Redis workload (RUNS=3, Appendix)

See Table 10. hz3 was fastest in 3 out of 5 patterns.

4.10 Latest Snapshot (2026-02-18, Addendum)

After the main body was drafted, we re-ran a 4-allocator snapshot (hz3/hz4/mimalloc/tcmalloc) with higher-run medians: MT lane x remote% matrix (`RUNS=10`) and redis-like real workload (`RUNS=10`). The split between hz3 and hz4 is clearer in this refreshed dataset.

- `main_r0` (local-heavy): hz3 leads (375.4M ops/s)
- `main_r90` (remote-heavy): hz4 leads (67.6M ops/s)
- `cross128_r90` (high-thread remote): hz4 leads by a large margin (50.65M ops/s)
- redis-like (memtier 15s, RUNS=10): hz3 571,199 / mimalloc 568,740 / tcmalloc 568,052 / hz4 560,576

Operationally, the most stable policy at this point is: default to hz3, and switch to hz4 for remote-heavy/high-thread cases.

Also, hz4 previously had a redis preload segfault (`rc=139`), but after the `malloc_usable_size` interpose fix, redis-like `RUNS=10` completed with `n_ok=10`.

4.10.1 Minimal Profile Selection Rules

From the refreshed `RUNS=10` results, the practical mapping is:

Workload shape	Recommended profile	Evidence (2026-02-18)
local-heavy (`main_r0`)	`hz3`	375.4M (hz4 137.4M)
remote-heavy (`main_r90`)	`hz4`	67.6M (hz3 62.6M)
high-thread remote (`cross128_r90`)	`hz4`	50.65M (hz3 1.80M)

4.11 Experimental Conditions

- CPU: 16-core x86_64

- OS: Ubuntu (native)
- Compiler: GCC (system default), optimization as per build default
- Build Configuration: hz3 scale lane / hz4 default
- Number of Runs: RUNS=10 (median adopted); some legacy appendix data remains RUNS=3/5
- Comparison Targets: mimalloc, tcmalloc (gperftools), system (glibc)

Reproduction Steps:

- Follow the procedures in `docs/benchmarks/2026-01-24_PAPER_BENCH_WORK_ORDER.md`. (Includes SSOT / T sweep / R sweep / dist_app / RSS / Appendix)

Logs are saved in `/tmp/hz3_ssot_YYYYMMDD_HHMMSS/` and include: - `[BENCH-HEADER]`: git commit, build flags, execution parameters - Median, min, max for each run - SSOT statistics at atexit (alloc/free counts, refill/drain counts, etc.)

5. Discussion

5.1 Superiority in Remote-Heavy Conditions

At R=90 (T=16), hz4 is the fastest, and the remote-heavy optimization via message-passing functions effectively. On the other hand, at R=0, hz3 is the fastest, indicating that fixed costs in local-only are small. This is due to the following designs:

1. **Owner Stash (S44)**: Intermediate buffer during remote free avoids mutex contention.
2. **RemoteStash Ring (S41)**: Sparse structure (TLS 6KB) maintains cache efficiency.
3. **S112 Full Drain Exchange**: Removes retry loops via bulk reclamation with atomic exchange.

5.2 Performance in Real Applications

- **ST dist_app**: tcmalloc is fastest, hz3 follows closely.
- **Larson sanity**: hz3 is at a level equivalent to tcmalloc.

From these results, it was confirmed that hz3 is at a competitive level even in real workloads.

On the other hand, in the mimalloc-bench subset, winners and losers vary by benchmark, and the trade-off between speed and memory efficiency appeared clearly.

In S44 ablation, disabling OwnerStash caused throughput to collapse by 93.5% and L1 misses to increase by over 10x. S44 is not a reduction target but a “load-bearing” optimization supporting performance.

5.3 Scaling Challenges

hz4 scales strongly at T=16, but remains weaker in ST and local-heavy conditions. This is not simply a defect; it reflects a deliberate remote-specialized trade-off. At present, usage-based profile separation (hz3/hz4) is the pragmatic choice, while integrated optimization remains future work.

- Continuation of P95-C series (notify queue/mailbox shard)
- Further reduction of central bin contention

5.4 Lessons Learned from NO-GO Cases

We recorded over 20 NO-GO cases and learned the following lessons:

1. **"Obviously fast" is denied by actual measurement** (S110-1 SegMath): Even with 100% hit rate, it loses due to boundary costs.
2. **Reliance on synthetic is dangerous** (S121 Series): Improvement in xmalloc-test but regression in other MT.
3. **Branch prediction and fixed cost shape are dominant** (S128 Defer Minrun): Regression due to increased branch-misses in hash probe.
4. **PGO depends on profile** (S143): Improvement in r0/r50 but regression in r90.
5. **Excessive prefetch causes regression** (S155): early prefetch dist=2 regressed -6~7% in r90/r50.

5.5 Practical Effect of Box Theory

- **Boundary Concentration**: Clarified the impact scope of optimization by aggregating transition points to the refill path.
- **Reversibility**: Enable A/B switching with compile-time flags, allowing recording and reproduction of over 20 NO-GO cases.
- **Observability**: Completely record build/execution conditions via atexit one-shot logs (SSOT).
- **Fail-Fast**: Early detection of UAF/double-free via PTAG32 tag==0 detection and central bin list corruption detection.

5.6 Operational Guidance (Current)

- Default: `hz3` (local-heavy, redis-like, lower-RSS oriented)
- Switch: `hz4` (remote-heavy, high-thread cross cases)
- R&D direction: hz4 retries should prioritize big-box structural changes over additional micro-boxes

6. Limitations and Future Work

- This paper primarily adopts results from Ubuntu (native).
- PTAG32 routing simplifies free branching but tends to **increase L1 misses** due to tag referencing. A trade-off with the header method remains, so build switching is performed according to usage.
- Fragmentation is evaluated by observation of RSS time series and retention amount, but strict modeling of fragmentation ratio is a future challenge.
- Some benchmarks (Redis, etc.) have strong internal allocators, tending to shrink differences.
- Future challenges are (1) improving scaling under high-thread/remote-heavy conditions, and (2) enabling learning layers (ACE/ELO/CAP) and establishing a safe operation mode.

7. Conclusion

Box-Theory-based hz3 achieved top speed and minimum RSS in local-only conditions. In addition, by separating hz4 as a remote-specialized profile, we reached practical performance in high-thread remote regimes that are hard to cover with a single profile. Going forward, we keep dual-profile operation and investigate integrated optimization through larger structural boxes.

Note (Origin of Name)

"hakozuna" originates from Box Theory born in the separate project "hakorune," and "zuna" means the rope of a Yokozuna. It was adopted as a name symbolizing the design philosophy and goal (strength and stability).

References

[1] gperftools (tcmalloc). https://github.com/gperftools/gperftools
[2] jemalloc. https://github.com/jemalloc/jemalloc
[3] mimalloc. https://github.com/microsoft/mimalloc
[4] google/tcmalloc. https://github.com/google/tcmalloc

Artifacts/Code

https://github.com/hakorune/hakozuna https://zenodo.org/records/18674502 https://doi.org/10.5281/zenodo.18674502

Acknowledgements

Writing of this paper was assisted by Claude Code and ChatGPT / ChatGPT Pro. Implementation work was primarily assisted by Claude Code.

Table 2: SSOT (T=16, R=90, RUNS=10, median; M ops/s).

Allocator	hz3	hz4	mimalloc	tcmalloc
ops/s	71.6	81.7	72.2	77.8

Table 3: T sweep (R=90, RUNS=3, median; M ops/s).

T	hz3	hz4	mimalloc	tcmalloc
4	83.1	101.6	75.6	78.4
8	80.8	73.2	75.9	66.7
16	76.6	106.3	85.0	79.5

Table 4: R sweep (T=16, RUNS=3, median; M ops/s).

R	hz3	hz4	mimalloc	tcmalloc
0%	359.6	249.6	300.5	357.0
50%	94.4	97.4	84.6	103.3
90%	75.3	97.3	75.1	74.9

Table 5: ST dist_app (RUNS=3, median; M ops/s).

Allocator	ops/s
tcmalloc	80.2
hz3	75.2
mimalloc	73.4
hz4	51.6

Table 6: RSS MT remote (T=16, R=90; GB, lower is better).

Allocator	Max RSS (GB)
hz3	1.36
mimalloc	1.52
hz4	2.04
tcmalloc	2.34

Table 7: RSS ST dist_app (KB, lower is better).

Allocator	Max RSS (KB)
hz3	3456
mimalloc	4480
tcmalloc	7936
hz4	14464

Table 8: Memory-release tuning (SSOT T=16, R=90, RUNS=10, median; M ops/s / GB).

Allocator (mode)	ops/s	RSS (GB)
hz3 (S64=0)	115.2	1.39
hz3 (S64 ON)	111.7	1.48
mimalloc (baseline)	102.1	1.12
mimalloc (purge=0)	112.1	0.52
tcmalloc (baseline)	63.8	2.79
tcmalloc (release_rate=10)	62.9	2.85
jemalloc (decay=0/0)	106.9	0.12

Table 9: mimalloc-bench subset (RUNS=5, median; ops/s).

Benchmark	hz3	hz4	mimalloc	tcmalloc
alloc-testN	24064	89856	13952	17664
xmalloc-testN	98048	71424	92232	44028
sh8benchN	239268	161088	159152	128588
cache-scratchN	9856	4600	3712	7680

Table 10: Redis workload (RUNS=3, median; M ops/s).

Pattern	hz3	hz4	mimalloc	tcmalloc
SET_ADD	2.59	2.22	2.46	2.31
GET	2.75	1.89	2.43	2.23
LPUSH	2.48	2.40	2.59	2.15
LPOP	2.42	2.15	2.59	2.07
RANDOM	2.32	1.93	2.20	2.22