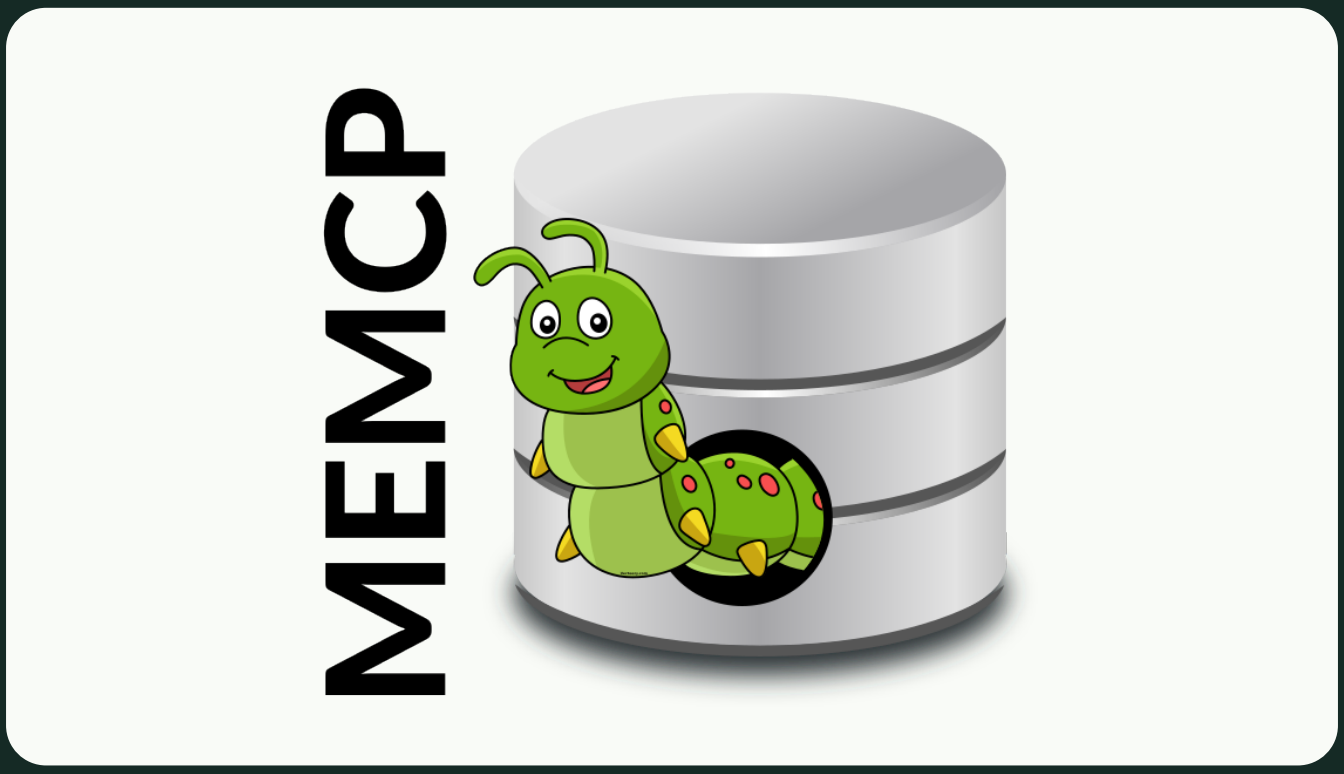
MEMCP / PERFORMANCE & EDGE WHITEPAPER

Keep your application. Make slow SQL fast.

A migration path for MariaDB and PostgreSQL workloads with fast search, live analytics and selective durability.



One workload. One comparison. Your numbers.

Send a representative slow query and data shape to info@launix.de

Start with the query that hurts

Measured application-shaped queries show where MemCP can change the economics of an existing SQL application. These are workload results, not a universal database speedup.

36.0×

WordPress search + count

30.405 ms MariaDB → 0.844 ms MemCP-JIT [1]

21.1×

Comment count with join

3.240 ms MariaDB → 0.153 ms MemCP-JIT [1]

6.2×

Monthly archive query

0.718 ms MariaDB → 0.116 ms MemCP-JIT [1]

Even small queries can benefit

In the same WordPress fixture, a one-row option lookup fell from 0.079 ms to 0.056 ms with MemCP-JIT: about 29% lower latency for that query. A separate in-process PHP PDO path measured 11.57 µs for SELECT 1, versus 37.19 µs through a local MySQL TCP connection; this is a transport comparison, not an equivalent database-engine comparison. [1]

A second route: PostgreSQL search

In one project-reported filtered-list workflow over roughly one million documents, the same query took about 30 seconds on PostgreSQL and 1.6 seconds on MemCP. The published summary does not make that ratio portable to other schemas or hardware; it identifies a query shape worth testing. [2]

WHAT A BUYER GETS

Faster filtered search and reporting can remove a visible customer delay and reduce the need for separate read replicas or analytics pipelines. The relevant metric is the complete application request under its real concurrency and correctness constraints.

Scope of the evidence. The seven WordPress query shapes used one client and a specified snapshot; the two metadata queries returned zero rows. Other isolated OLTP measurements in the project documentation have taken 1.3–2.0× as long on MemCP. Re-run your own workload, including writes and end-to-end requests. [1, 2]

02 / MIGRATION PATH

Change the engine, keep the workflow

MemCP is designed for existing SQL applications. The migration path differs between MariaDB/MySQL and PostgreSQL, and every production cutover needs compatibility checks.

MariaDB / MySQL: familiar client connection

MemCP speaks the MySQL wire protocol, so an application can often test it with its existing MySQL connector by changing the endpoint and importing data. PDO/MySQL and supported client tools can connect without a new query API. Verify the application's actual SQL, metadata expectations, transactions and driver behavior before calling a specific deployment "drop-in." [3, 4]

SMALL FIRST EXPERIMENT

Copy one database → point a staging client to MemCP's MySQL port → replay the slow query → compare result rows and end-to-end latency.

PostgreSQL: import and adapt the connection

MemCP can import from a live PostgreSQL connection or supported dumps, and its PostgreSQL-oriented SQL frontend is available over HTTP. It does not provide a PostgreSQL wire-protocol server: an unmodified libpq connection cannot simply point to MemCP. Treat this as a migration with query and client validation, then test the same user-visible workload. [3]

The architecture behind the speed

MemCP combines compressed, column-oriented main storage with a write-friendly delta. Search, filtering and aggregation can operate over relevant columns while the application continues updating operational data. Hot data is memory-oriented; cold persistent data may reload from storage. JIT is an optional experimental path requiring a patched Go toolchain. [4, 5]

- Good first candidates: slow filtered lists, count queries, reporting on live data, search-heavy application pages.
- Cutover gates: result parity, supported SQL and client behavior, sustained write load, backup/restore, restart recovery and rollback.

Pay for durability where it matters

A table-level storage choice lets one application protect irreplaceable records and give high-volume, replaceable data a faster, less write-intensive path.

SAFE

Protect the records you cannot lose

WAL synchronized at commit; one early run recorded ~1,700 individually synchronized writes/s. That historical number is not a fixed ceiling: batching and storage change it.

Fit: Orders, configuration, permissions

LOGGED

Trade power-loss exposure for write rate

WAL without fsync; a project measurement reached ~10× the same safe write path. Process-crash recovery is possible, but recent writes can vanish after power loss.

Fit: Replaceable high-rate updates

SLOPPY

Stop writing every sample to WAL

No WAL; a compressed main generation is normally published every 15 minutes. Useful for flash-backed telemetry, but an unclean stop loses the delta since the last successful rebuild.

Fit: Sensors, usage statistics, staging

MEMORY / CACHE

Use RAM for temporary state

No row persistence; cache can also disappear under memory pressure. Remove write traffic for data the application can recreate.

Fit: Sessions and derived caches

Flash claim: fewer continuous database writes can reduce wear; the 15-minute interval is a normal schedule, not a guaranteed maximum loss window. A rebuild also writes data. Measure physical device writes, retention and recovery on the target SD/eMMC hardware before quoting lifetime gains. [5]

04 / WORK WITH US

Put your slowest query on trial

MemCP is open source. LAUNIX’s proposed monthly performance service combines tuned operation with continuing query and schema optimization.

A practical, low-friction pilot

• 1. Send the pain point. Share a slow query, schema, row counts, current timing and the acceptable data-loss window. Use sanitized data or a reproducible generator.

• 2. Reproduce the workload. Run MariaDB or PostgreSQL and MemCP against equivalent results; record hardware, versions, client path, concurrency and durability.

• 3. Decide from outcomes. Compare full request latency, p95/p99, CPU/RAM, physical writes and recovery behavior. Keep a rollback route for any cutover.

REQUEST A WORKLOAD ASSESSMENT

info@launix.de

Include one representative query and its current timing.

Project and documentation: memcp.org | Source: github.com/launix-de/memcp

What ongoing optimization covers

A monthly engagement can cover CPU/JIT build selection, slow-query analysis, schema and storage-mode choices, monitoring, backup and recovery exercises, and regression checks. Agree on target outcomes and reporting cadence in the pilot; no blanket “2–3×” platform improvement is promised without a workload-specific test.

EVIDENCE & LIMITS

[1] memcp.org/wiki/Benchmark_MemCP_vs._MariaDB_on_Wordpress — snapshot 43d0c8140; medians and parity checks.

[2] memcp.org/wiki/Performance_Measurement — PostgreSQL filtered-list example and isolated OLTP caveat.

[3] memcp.org/wiki/Migration_from_MySQL_and_PostgreSQL — import, protocols and migration checklist.

[4] github.com/launix-de/memcp — project status, architecture, JIT setup and source.

[5] memcp.org/wiki/Persistency_and_Performance_Guarantees — modes, historical write tests and failure semantics.

All quoted results come from project-published measurements and may change with release, hardware and workload. This document proposes a customer pilot and service offering; it does not report an independent benchmark or a measured SD-card lifetime.