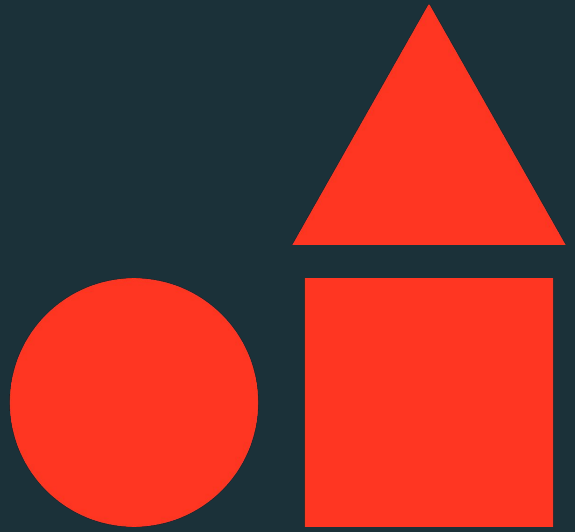
databricks

Session memory footprint

Ways to reduce the memory footprint of connections

Matthias van de Meent
2026-05-21



About me

Matthias van de Meent

Professional path

2017-'21: Cofano

FE → Full stack → Infra/DBA

2021-'25: Neon

PG hacker & software engineer

2025-now: Databricks

PG hacker & software engineer

PostgreSQL hacker of 5 years, mostly performance oriented

- Help out parallel index creation (BRIN, GIN)
- Heap page savings
- COPY progress reporting
- ANALYZE (SERIALIZE) option
- etc.

Agenda

Today

Relcache

- Contents
- Optimizations
- Deduplications

Memory context optimizations

Not today

Planner/executor memory usage

Shared memory (pgconf.dev 2025)

- Shared buffers, lock tables, etc.

On-disk data (pgconf.dev 2024)

- Layouts of pages, catalog tuples, etc.

Relcache contents

- Primary entrypoint for retrieving relation data
  - tables, views, indexes, etc.
- Various allocated data structures, both internal and external:
  - RelationData
  - ~~IndexAmRoutine~~ *(static const * as of PG19)*
  - TupleDesc
  - hashmaps for lookups
  - etc.
- *consumes over 50% of memctx-managed memory in an idle connection*

Relcache optimization

RelationData: Many relation types

- fields: shared vs. owned
  `TupleDesc rd_att` (any relation) vs. `IndexAmRoutine *rd_indam` (indexes)

Alias memory for owned data:

- Index-specific fields (128B)
- Fields never used by indexes (>128B)

Relcache optimization

RelationData: Many relation types

- fields: shared vs. owned
  `TupleDesc rd_att` (any relation) vs. `IndexAmRoutine *rd_indam` (indexes)

Alias memory for owned data:

- Index-specific fields (128B)
- Fields never used by indexes (>128B)

… But won't save memory when allocated in ASET (488 → 360 bytes; both use 512 B bucket in ASET allocator)

Relcache deduplication

Frequent pattern: ~equivalent object definitions

Duplication can be caused by e.g.

- Partitioning
- Shared definitions (indexes)
  - *CREATE INDEX ON tableA USING btree (id int8_ops)*
  - *CREATE INDEX ON tableB USING btree ("count" int8_ops)*

Relcache deduplication

Deduplicate large/frequently duplicated structures

- RelationData's index opclass-related allocations
- TupleDesc's attribute data

Relcache deduplication

Deduplicate RelationData's fields populated from pg_index:

pg_index.indam

PG19's bc6374cd7 "Change IndexAmRoutines to be statically-alocated structs"

- `IndexAmRoutine *.rd_indam`

Relcache deduplication

Deduplicate RelationData's fields populated from pg_index:

pg_index.indnkeyatts + indclass + indoption

- `rd_opfamily`/`rd_opcintype`/`rd_support`/`rd_supportinfo`/`rd_opcoptions`
- None of these contain table/index references; only "shape" info
- Fields trivial to deduplicate

  WIP branch at https://github.com/MMeent/postgres/tree/relcache-dedup-shapes

Relcache deduplication

Deduplicate RelationData's fields populated from pg_index:

pg_index.indkey, .indexprs, .indpred

- rd_indexprs/rd_indpred
- Contain table references = more entropy & less duplicated definitions
- Partitioned workloads may still make this worthwhile, but I haven't spent time exploring this further

Relcache deduplication

Deduplicate TupleDesc

With partitions, the TupleDescs of the partitions are all the same, right?

(or at least, close enough)

Relcache deduplication

Deduplicate TupleDesc

With partitions, the TupleDescs of the partitions are all the same, right?

Relcache deduplication

Deduplicate TupleDesc – differences

- Different column numbers
  - Dropped columns
  - Columns added to partitions
  - Manually attached partitions with different column order
- Different column options
  - e.g. statistics targets may be configured differently

Relcache deduplication

Deduplicate TupleDesc - differences (cont.)

- TupleDesc contains pg_attribute verbatim
  - ***pg_attribute.attrelid is always different for each table***, and so can't be deduplicated across tables

Relcache deduplication

Deduplicate TupleDesc

Split TupleDesc definition:

- Shared/deduplicated shape:
  - atttype/attlen/attalign/attbyval/attisdropped/attnullability/etc
  - Generally useful for both indexes and normal partitioning schemes
  - C.F. CompactAttribute
- Shared/deduplicated column name list:
  - Saves N times NAME_DATA_LEN in case of normal partitioning patterns
- Private relid list
  - Relids are (presumably) needed for planner TupleDescs

Relcache deduplication

Deduplicate TupleDesc

Critically: **Removes FormData_pg_attribute[] from catcache memory**

- Decouples one more direct reliance on NAME_DATA_LEN for struct layouts
- (far future) Could enable longer attribute/column names

Memory context selection

using a better allocator for long-lived allocations

- catcache doesn't care as much about low-latency as it would about resource usage
- aset is not great for catcache:
  - Overallocates memory (up to 100%)
  - Oversized initial memctx allocation (1kB)

Memory context selection

A better allocator for long-lived allocations

Andres' suggestion: proxy.c allocator

- Proxies memory requests into malloc
- Tracks individual allocations in the memctx

Memory context selection

Pro

- No more overallocated memory for memctx blocks
- Better sized allocations

Contra

- Takes away some control on which memory is kept around to avoid OS (re)alloc overheads
- Assumes malloc is a better allocator than what we can do
  (probably correct for aset, but what about others?)

Memory context selection

A better allocator for long-lived allocations

Alternative approaches

- Proxy context, which proxies allocations into other contexts
  - Removes the need for memory blocks/allocator
- New context better optimized for long-lived allocations
  - Don't care about low-latency (de)allocation patterns (aset.c)
  - Instead care about reducing memory wastage
  - Implement jmalloc/glibc's malloc/etc. ideas in new MCTX implementations?
- CTA: there are 7 MCTX_UNUSED IDs. Try implementing one and see what works.


databricks