NEON

Multi-threaded PostgeSQL?

Heikki Linnakangas

NEON

Why now?

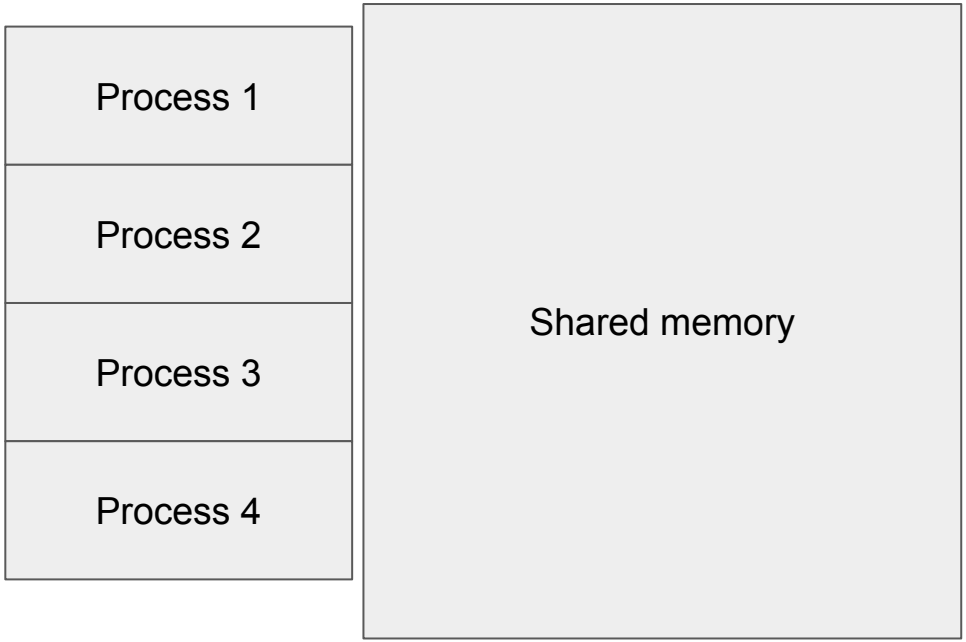
- Started as a hallway conversation at PGCon 2023
- Followed up on the mailing list:
  https://www.postgresql.org/message-id/31cc6df9-53fe-3cd9-af5b-ac0d801163f4@iki.fi

Status

- There is no patch
  - Some preliminary refactoring at Refactoring backend fork+exec code thread
  - Some work on annotating global variables at https://github.com/hlinnaka/postgres/tree/threading
- I might work on this, or not. No promises!

NEON

Current multi-process architecture



What's in shared memory? (1 / 3)

```
/*
 * Size of the Postgres shared-memory block is estimated via moderately-
 * accurate estimates for the big hogs, plus 100K for the stuff that's too
 * small to bother with estimating.
 *
 * We take some care to ensure that the total size request doesn't
 * overflow size_t.  If this gets through, we don't need to be so careful
 * during the actual allocation phase.
 */
size = 100000;
size = add_size(size, PGSemaphoreShmemSize(numSemas));
size = add_size(size, SpinlockSemaSize());
size = add_size(size, hash_estimate_size(SHMEM_INDEX_SIZE,sizeof(ShmemIndexEnt)));
size = add_size(size, dsm_estimate_size());
size = add_size(size, BufferShmemSize());
size = add_size(size, LockShmemSize());
size = add_size(size, PredicateLockShmemSize());
size = add_size(size, ProcGlobalShmemSize());
size = add_size(size, XLogPrefetchShmemSize());
size = add_size(size, VarsupShmemSize());
size = add_size(size, XLOGShmemSize());
size = add_size(size, XLogRecoveryShmemSize());
size = add_size(size, CLOGShmemSize());
size = add_size(size, CommitTsShmemSize());
size = add_size(size, SUBTRANSShmemSize());
size = add_size(size, TwoPhaseShmemSize());
```

What's in shared memory? (2 / 3)

```
    size = add_size(size, BackgroundWorkerShmemSize());
    size = add_size(size, MultiXactShmemSize());
    size = add_size(size, LWLockShmemSize());
    size = add_size(size, ProcArrayShmemSize());
    size = add_size(size, BackendStatusShmemSize());
    size = add_size(size, SInvalShmemSize());
    size = add_size(size, PMSignalShmemSize());
    size = add_size(size, ProcSignalShmemSize());
    size = add_size(size, CheckpointerShmemSize());
    size = add_size(size, AutoVacuumShmemSize());
    size = add_size(size, ReplicationSlotsShmemSize());
    size = add_size(size, ReplicationOriginShmemSize());
    size = add_size(size, WalSndShmemSize());
    size = add_size(size, WalRcvShmemSize());
    size = add_size(size, PgArchShmemSize());
    size = add_size(size, ApplyLauncherShmemSize());
    size = add_size(size, BTreeShmemSize());
    size = add_size(size, SyncScanShmemSize());
    size = add_size(size, AsyncShmemSize());
    size = add_size(size, StatsShmemSize());
    size = add_size(size, WaitEventExtensionShmemSize());
#ifdef EXEC_BACKEND
    size = add_size(size, ShmemBackendArraySize());
#endif

    /* include additional requested shmem from preload libraries */
    size = add_size(size, total_addin_request);
```

NEON

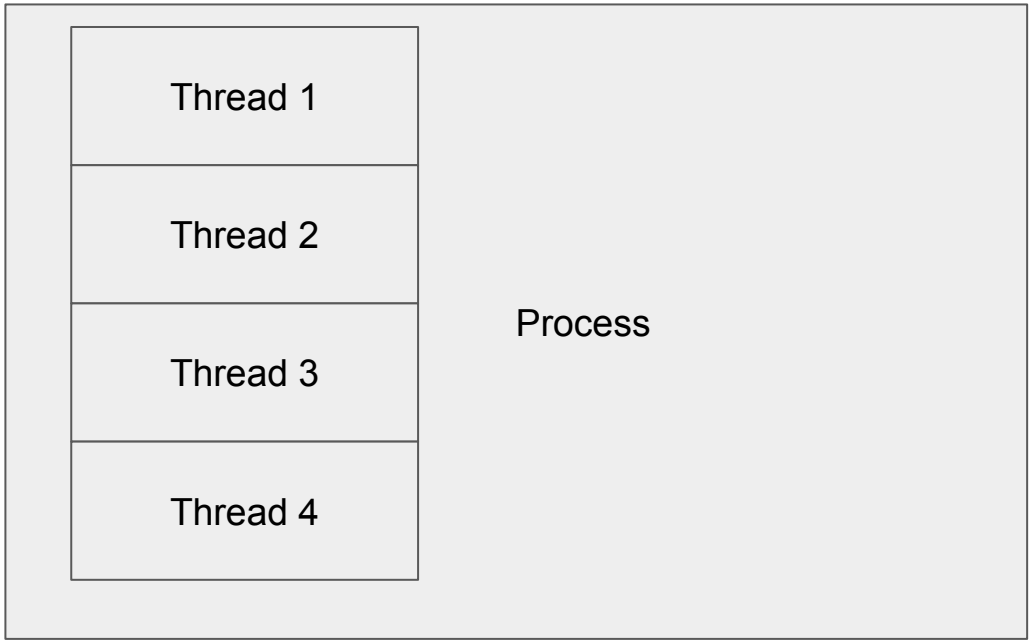
What’s in shared memory? (3 / 3)

Dynamic Shared Memory for communication between parallel workers:

- Parallel sort state
- hash table for hash joins
- sharing record type cache
- Logical replication workers
- pgstat

Multi-threaded architecture

- Thread per connection



What’s the big difference?

	Process-per-connection	Thread-per-connection
Address space	Per-process	Shared
Shared data structures	Tedious	Easy

NEON

Benefits, immediate

Performance:

- Fewer TLB misses, maybe
- Less page table overhead

NEON

Benefits, long-term

Makes developing these things easier:

- Cheaper connections, built-in “connection pool”
- Shared relcache, plan cache
- Resizing fixed-size shared memory areas
  - Shared_buffers, max_locks_per_transactions etc.
- Changing settings without restart
- Track snapshots that are in use to vacuum more aggressively
- EXPLAIN ANALYZE on the fly
- Limiting memory usage per session / connection

NEON

Objections

- **It’s not worth the effort**
- **Too much incompatibility, think of extensions. python 2 vs 3**
- **It will introduce lots of bugs**
- Multi-process gives better isolation

NEON

Objection: It’s not worth the effort

If that’s true, then it won’t happen

Wrong question to ask:

> This is open source, people spend their time on what *they* decide is worth it

NEON

Objection: Too much incompatibility

- We don’t want a python 2 vs python 3 situation with the ecosystem
- Even within the core project, there should be no massive disruption to how to you deploy and administer

Objection: It will introduce lots of bugs

> For the record, I think this will be a disaster. There is far too much code that will get broken, largely silently, and much of it is not under our control.
>
> regards, tom lane

- I hope not!
- Need a beta period

NEON

Objections

- It’s not worth the effort
- Too destabilizing, think of extensions. python 2 vs 3
- It will introduce lots of bugs
- **Multi-process gives better isolation**

Objection: Multi process gives better isolation

- Chrome uses a process per tab for isolation

In PostgreSQL, it doesn't give as much isolation as you might hope:

- If one process crashes, all other processes are killed
- Stomping over shared memory can already cause corruption-at-a-distance

- Multiple processes are easier to work with in debugger, strace, top etc.

NEON

Objection: Memory leaks will be worse

- We’re pretty good at not leaking resources. MemoryContexts and ResourceOwners work great.
- We only recently fixes a session-lifetime leak in LLVM

NEON

Multi-process enforces discipline

- Multi-process architecture forces *discipline* on data structures that are shared across processes
- It forces the discipline by making it so painful that you don’t want to do it.
- There are other ways, like naming conventions to enforce discipline

NEON

Previous attempts

- Early Windows port
- https://github.com/cmu-db/peloton/wiki/Postgres-Modifications (2015)
- https://github.com/postgrespro/postgresql.pthreads (2018)

Other projects that have made the switch

- Apache2
  - MPM (Multi-Processing Modules), *prefork*, *worker*, or *event*
- Oracle
- Firebird

NEON

Here’s the plan!

NEON

The Plan

- For each connection, launch thread instead of process
- Annotate all global variables
- Add flags for extensions to declare if they’re thread-safe
- Rewrite some subsystems

Session local state

- Currently in global variables
- Convert to thread-local variables
- Or gather them all to a Session struct

14 src/backend/access/table/tableam.c

```
@@ -45,8 +45,8 @@
45  45  #define PARALLEL_SEQSCAN_MAX_CHUNK_SIZE        8192
46  46
47  47  /* GUC variables */
48      - char       *default_table_access_method = DEFAULT_TABLE_ACCESS_METHOD;
49      - bool        synchronize_seqscans = true;
    48  + session_guc char       *default_table_access_method = DEFAULT_TABLE_ACCESS_METHOD;
    49  + session_guc bool        synchronize_seqscans = true;
50  50
51  51
52  52  /* ----------------------------------------------------------------------------
```

NEON

Extensions

- Extensions need a transition period, independently of PostgreSQL
- Add a flag to control file:
  - Requires processes
  - Requires threads
  - Works in either model
- Need tools for checking for re-entrant
  - Static analysis tool to catch global variables
  - Tests with concurrency

NEON

Rewrite some subsystems

Some subsystems will need to work differently in multi-threaded model:

- Virtual file descriptors (fd.c)
- Inter-process signals (SIGUSR1, SIGHUP etc)
- Launching new connections
- Postmaster restart_on_crash

NEON

Transition period

- First version will be buggy
- Extensions need time to catch up
- Need a transition period, where you can choose with a GUC
  - 2 years? 5 years?
- IMHO the goal has to be to eventually remove the multi-process mode, or this is not worth it

NEON

Then what?

NEON

Reap the benefits

- Simpler parallel worker IPC
  - Get rid of DSM, DSA, replica with plain palloc()s + lwlocks
- Share Relcache, catcaches, plan caches
- Allocate temp buffers more flexibly (or move to shared buffers)

NEON

Beyond thread-per-connection

- Thread pools, queuing
- Thread per core
- Shard per core (ScyllaDB)
- Async execution
  - Have some of this in FDWs already
  - Would help to parallelize I/O in more places

TODO

- Global variables
- Extensions
- Transition period
- PIDs in user-facing APIs (pg_terminate_backend(<pid>), query cancellation message in client protocol)
- Signals between backend processes
- setlocale() -> uselocale()
- python is single-threaded

NEON

Thank you!

Q & A

pgsql-hackers thread:
https://www.postgresql.org/message-id/31cc6df9-53fe-3cd9-af5b-ac0d801163f4@iki.fi