HOW 2026
Hello Open-source World

credativ

Linux and PostgreSQL in the Multiverse of Connections

Why Are PostgreSQL Connections Expensive?

Josef Machytka <josef.machytka@credativ.de>

2026-04-28 HOW2026: PostgreSQL & IvorySQL Eco Conference

credativ GmbH

HOW 2026
Hello Open-source World

credativ

- Founded 1999 in Jülich, Germany
- Close ties to Open-Source Community
- More than 40 Open-Source experts
- Consulting, development, training, support (3rd-level / 24x7)
- Open-Source infrastructure with Linux, Kubernetes, Proxmox
- Open-Source databases with PostgreSQL
- DevSecOps with Ansible, Puppet, Terraform and others
- Since 2025 independent owner-managed company again

credativ.de

Josef Machytka

HOW 2026 Hello Open-source World

credativ

- Professional Service Consultant - PostgreSQL specialist at credativ GmbH
- 33+ years of experience with different databases
- PostgreSQL (13y), BigQuery (7y), Oracle (15y), MySQL (12y), Elasticsearch (5y), MS SQL (5y)
- 10+ years of experience with Data Ingestion pipelines, Data Analysis, Data Lake / Warehouse
- 3+ years of practical experience with LLMs / AI / ML including architecture and principles
- From Czechia, living now 12 years in Berlin

- LinkedIn: linkedin.com/in/josef-machytka
- Medium: medium.com/@josef.machytka
- YouTube: youtube.com/@JosefMachytka

- GitHub: github.com/josmac69/conferences_slides
- ResearchGate: researchgate.net/profile/Josef-Machytka

All My Slides:

Recoded talks:

HOW 2026
Hello Open-source World

Acknowledgement

credativ

Many thanks to the IvorySQL community for supporting my participation in the HOW2026 conference

Many thanks to the Highgo company for the opportunity to test on a Phytium server

Table of Contents

Talk based on Linux 6.x, PostgreSQL 18, and x86_64 architecture

NUMA related facts are relevant mostly for the ARM64 Phytium server, Kylin Linux v10

Diagrams and pictures created by Claude Opus 4.7 and Gemini 3.1 Pro

HOW 2026
Hello Open-source World

credativ

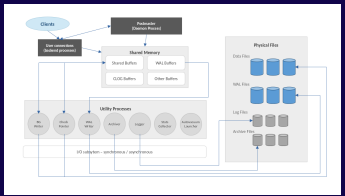
PostgreSQL Multi-Process Architecture

PostgreSQL Multi-Process Architecture

HOW 2026 Hello Open-source World

credativ

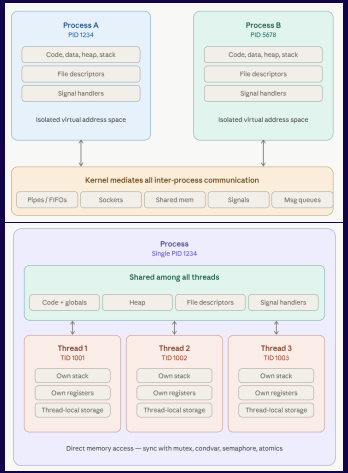
- PostgreSQL uses a multi-process architecture
  - Every connection, every background task is a separate OS process
  - Leads to expensive CPU, memory, and context-switching overhead
  - Inter-process communication (IPC) is necessary via shared memory



- High connection counts (>300–400) degrade performance
  - Too many context switches between processes
  - Too many Translation Lookaside Buffer (TLB) misses
  - Too many resources consumed by idle connections
  - Exponential performance degradation

PostgreSQL Connections

- Each connection is an independent process
- Processes chosen for higher stability in the 1990s
- Reliable threads implemented in Linux 2.6 (2003/2004)
- Main idea: processes are isolated, heavy, but stable
  - Each process has its own isolated memory
  - Great for stability, security, but resource-intensive
  - Starting a new process incurs some overhead
  - But forking has been optimized lately
- Ongoing discussion about switching to threads
  - Threads are now reliable and considered lightweight
  - But handling memory safety depends on developers



Shared Memory on Linux

- Linux implements shared memory for interprocess communication
- Linux philosophy: `Everything is a file`
  - -> `dentry` (directory entry) & `inode` structures
- Shared memory on Linux implemented via `tmpfs` filesystem
  - Filesystem interface to access memory as files
  - Introduced in Linux kernel 2.4 (2001)
  - Successor of older `ramfs`
  - Internally used even if tmpfs is disabled for users
  - Can be used by SysV IPC (shmget, shmat) and mmap interfaces
- PostgreSQL originally used SysV shared memory segments
  - Since PG 9.3 POSIX shared memory via mmap is default on Linux

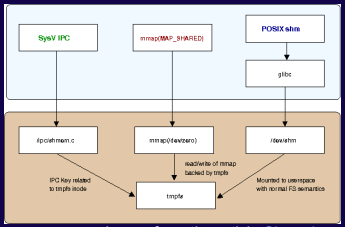


Image from the article Shared memory on Linux

Shared Memory on Linux

HOW 2026 Hello Open-source World

credativ

- Users interact with tmpfs via `/dev/shm` directory
  - This filesystem grows and shrinks dynamically as files are created or deleted
  - Size is limited by available RAM and swap space
  - Uses page cache - file I/O is done directly in memory
  - Writing allocates physical memory pages - associated with `dentry` and `inode` structures

- PostgreSQL uses /dev/shm for communication between processes
  - Has a small default on Docker (64MB), can be exhausted quickly - `--shm-size` / `shm_size` parameters
  - If /dev/shm is exhausted, PG reports "`could not resize shared memory segment`" error

```
my-linux ~ $ df -h | grep tmpfs
tmpfs                     3.2G  3.7M  3.1G   1% /run
tmpfs                      16G  543M   15G   4% /dev/shm
tmpfs                     5.0M  8.0K  5.0M   1% /run/lock
tmpfs                     3.2G  140K  3.2G   1% /run/user/1000
```

HOW 2026
Hello Open-source World

credativ

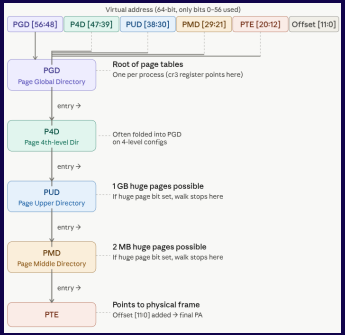
PostgreSQL Connections Memory Allocation

Linux Memory Allocation Management

HOW 2026 Hello Open-source World

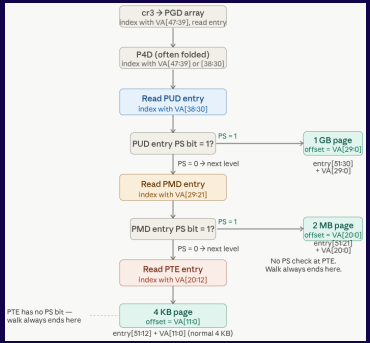
credativ

- Linux implements multiple levels of memory mapping tables (4 or 5)
- Mapping translates 64-bit virtual addresses to physical addresses
- Not all 64bits are used for mapping - 48 or 57 bits
  - Page global directory (PGD)
  - Page 4th level directory (P4D) - folded into PGD on 4-level configs
  - Page upper directory (PUD) - 1 GB pages walk ends here
  - Page middle directory (PMD) - 2 MB pages walk ends here
  - Page table entry (PTE) - standard 4 KB pages are stored here
- Tables contain pointers to the next level table or the final mapping
- PUD and PMD have PS bit set when huge pages are used



Standard Pages vs Huge Pages

- Linux fork() does not copy physical memory allocation
  - PTE small when connection starts - around 300 KB
  - Child process fills its tables on TLB Miss
- Shared buffer allocated in standard 4 KB pages
  - The whole hierarchy is traversed on each page access
  - Mapping is found on the last level (PTE)
- Shared buffer allocated in huge pages
  - Mapping of 1GB HugePage is found on PUD level
  - Mapping of 2MB HugePage is found on PMD level
- -> Huge pages significantly improve performance
- -> Reduce overall memory usage



PostgreSQL Connection Memory Allocation

HOW 2026 Hello Open-source World

credativ

- Shared buffers allocated as standard pages
  - 1 GB buffers in 4 KB pages = 2 MB PTE table
  - 8 GB buffers in 4 KB pages = 16 MB PTE table
  - 100 connections, 8 GB buffers = 1.6 GB PTE tables
  - 1000 connections, 8 GB buffers = 16 GB PTE tables (!!!)
- Shared buffers allocated as huge pages
  - 1 GB buffers in 2 MB huge pages = 4 KB PMD table
  - 8 GB buffers in 2 MB huge pages = 32 KB PMD table
  - 100 connections, 8 GB buffers in 2MB huge pages = 3.2 MB PMD tables
  - 1000 connections, 8 GB buffers in 2MB huge pages = 32 MB PMD tables
- Why Linux HugePages are Super Important for Database Servers

```
grep -E "PageTables" /proc/meminfo    ## all PTE tables together
grep -E "VmPTE" /proc/<PID>/status    ## connection PTE table
```

HOW 2026
Hello Open-source World

credativ

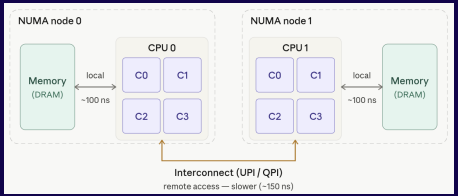
PostgresSQL and NUMA Architectures

PostgreSQL and NUMA Architectures

HOW 2026 Hello Open-source World

credativ

- NUMA (Non-Uniform Memory Access) architectures - divide the system into nodes
- Each node has its own CPU cores and local memory (not always attached to each node)
- Accessing local memory is faster than remote memory in other nodes
- Remote memory access causes higher latency -> NUMA penalty - 2x-3x slower
- Linux kernel offers NUMA balancing mechanisms
  - Automatic memory pages migration or migration of processes between nodes
  - `/proc/sys/kernel/numa_balancing` - 1/0 = on/off
  - Useful for general workloads, but what about PostgreSQL?

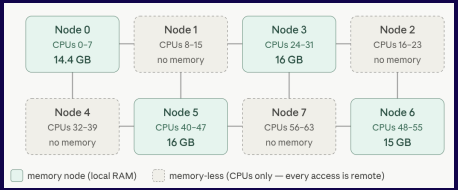


PostgreSQL on ARM64 Phytium

HOW 2026
Hello Open-source World

credativ

- Server: Phytium FT-2000+/64 - 64 cores ARM64, 62 GB RAM, 8 NUMA nodes
- Kylin Linux Advanced Server v10 (Halberd), kernel 4.19.90 (CentOS 8 lineage)
- Kernel re-spun by KylinSoft - added Chinese hardware support, security patches
- PostgreSQL-IvorySQL 18.3, 4 nodes with memory, 4 nodes without memory



target → / from ↓	0	1	2	3	4	5	6	7
0	10	20	40	30	20	30	50	40
1	20	10	30	20	30	20	40	30
2	40	30	10	20	50	40	20	30
3	30	20	20	10	40	30	30	20
4	20	30	50	40	10	20	40	30
5	30	20	40	30	20	10	30	20
6	50	40	20	30	40	30	10	20
7	40	30	30	20	30	20	20	10

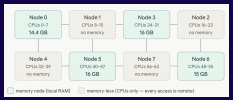
	Node 0 14,403 MB RAM present	Node 3 16,362 MB RAM present	Node 5 16,362 MB RAM present	Node 6 15,317 MB RAM present
Node 0 CPUs 0-7	10 local	30 +200%	30 +200%	50 +400%
Node 1 CPUs 8-15	20 +100%	20 +100%	20 +100%	40 +300%
Node 2 CPUs 16-23	40 +300%	20 +100%	40 +300%	20 +100%
Node 3 CPUs 24-31	30 +200%	10 local	30 +200%	30 +200%
Node 4 CPUs 32-39	20 +100%	40 +300%	20 +100%	40 +300%
Node 5 CPUs 40-47	30 +200%	30 +200%	10 local	30 +200%
Node 6 CPUs 48-55	50 +400%	30 +200%	30 +200%	10 local
Node 7 CPUs 56-63	40 +300%	20 +100%	20 +100%	20 +100%

Pinning PostgreSQL on NUMA nodes

HOW 2026
Hello Open-source World

credativ

- Forked processes inherit postmaster's bind policy - memory / CPUs
- Pinning both memory and CPUs to specific NUMA node(s)
  - PostgreSQL will use only specified node(s) for everything
  - Specified node(s) must have enough memory for everything
  - Shared buffers, other shared objects (WAL, locks), /dev/shm, connections
  - All processes running on specified node(s), jumping between cores on them
- Pinning just memory to specific NUMA node(s)
  - Processes will jump randomly between all nodes during their run
  - But their memory is pinned to specified node(s)
- Not pinning anything means completely random behavior

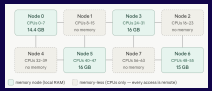


Pinning PostgreSQL on NUMA nodes

HOW 2026
Hello Open-source World

credativ

- Pinning memory / CPUs limits PostgreSQL resources to specific node(s)
  - Can be good for isolating workload on specific part of NUMA system
  - Use case for multiple PostgreSQL clusters on one NUMA system



- If size of shared buffers exceeds memory on specified node(s)
  - Buffers will be allocated on available memory, rest stays unmapped
  - Attempt to physically fill whole shared buffers crashes PostgreSQL

- I tested all variants of pinning/ not pinning on Phytium server
  - Differences in benchmarks were between 2 to 6 % and not consistent
  - For typical mainstream workloads pinning is useful only for isolation of PostgreSQL
  - NUMA latency was not significant; the distance table exaggerates

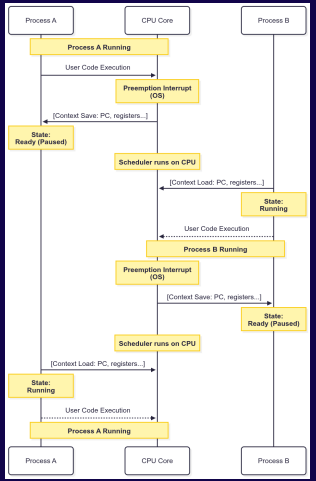
HOW 2026
Hello Open-source World

credativ

CPU and PostgreSQL Connections

Context Switching Overheads

- But how many ”active connections per core/thread”?
  - Depends on limitations of context switching mechanism
- Context switching brings overheads
  - APIC Interrupt Controller fires timer interrupt
  - APIC is integrated into each individual CPU core or thread
  - On interrupt, CPU saves current process state (context)
  - CPU loads new process state from memory
  - State includes CPU registers, program counter, stack pointer
  - TLB (Translation Lookaside Buffer) may need flush
  - -> TLB entries are newly tagged with process IDs

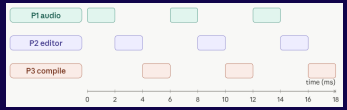


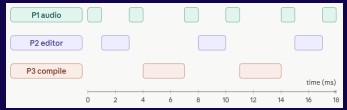
Linux CPU Schedulers

HOW 2026 Hello Open-source World

credativ

- Completely Fair Scheduler (CFS)
  - Original scheduler used in Linux 2.6+
  - ”Ideal multi-tasking” - N processes, 1/N speed
  - Every process gets an equal share of CPU time
  - In kernel 6.6+ replaced with EEVDF scheduler
- Earliest eligible virtual deadline first (EEVDF)
  - Newer scheduler, since October 2023
  - Introduced small, more frequent bursts of CPU time
  - Can prioritize interactive or latency critical tasks
  - But still should keep fair share of total CPU time
- Ubuntu uses 6.6+ from v24.04
- Debian 12 uses 6.1 -> Debian 13 jumps to 6.12
- RedHat Enterprise 9 uses 5.14 -> v 10 jumps to 6.12
- 6.12 contains many improvements and patches against 6.6+



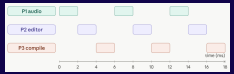


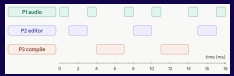
CFS vs EEVDF

HOW 2026
Hello Open-source World

credativ

- What switch from CFS to EEVDF means for PostgreSQL connections?
- Typical PG connections are often ”sleeping” - network, data fetch, lock wait
- Some connections are ”bursty” - many small transactions
- CFS can sometimes penalize ”bursty” workloads
  - Context switching storm by waking up many sleeping processes
  - Maintains ”average fairness” for all processes
  - Uniformity, no latency preference, only some preemption for ”sleepers”
- EEVDF can prioritize latency critical tasks
  - ”Bursty” workloads can show better latency - processed earlier
  - Waiting processes should not slow down bursty workloads
  - CPU-heavy workloads will not starve other processes
  - Most likely will improve mixed workloads
  - OLAP only workloads might slightly suffer more context switches
  - For OLAP we can use SCHED_BATCH policy for throughput





Connection Limits Recommendations

HOW 2026 Hello Open-source World

credativ

- Rule of thumb: 2 - 4 active OLTP connections per 1 CPU core
  - -> applies only for OLTP workloads on NVMe/SSD storage with huge pages enabled
  - -> sweet spot seems to be 3 active connections per 1 core
  - -> down to 1-2 active connections per 1 core for HDD storage or without huge pages
- OLAP workloads without parallelism -> 1 connection per 1 core
- OLAP workloads with parallelism -> 1 connection per [1+workers] cores
- -> Use PostgreSQL 14+ - allows linear scaling of MVCC
- -> Use PostgreSQL 18+ due to improved relation locking mechanism

HOW 2026
Hello Open-source World

credativ

How Much Memory PostgreSQL Connections Use?

PostgreSQL Connection Memory Usage

HOW 2026 Hello Open-source World

credativ

- 32GB RAM, PostgreSQL 18, shared_buffers=8GB, effective_cache_size=24GB, work_mem=64MB
- Numbers parsed by script from `/proc/PID/smaps` kernel interface

```
## output of top command
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 190747 postgres  20   0 8701512  20248  16876 S   0.0   0.1   0:00.00 postgres: postgres postgres 172.18.0.1(40278) idle

## python script output - smaps
Path                                                    Size       Rss       Pss  Pss_Dirty  Shr_Clean  Shr_Dirty  Prv_Clean  Prv_Dirty  Swap  SwapPss  Cnt
-------------------------------------------------------------------------------------------------------------------------------------------------------------
/usr/lib/postgresql/18/bin/postgres                     9296      4140      1156         75       3792        168        128         52     0        0    5
[anonymous]                                             1708       660       554        554          0        120          0        540     0        0   21
[heap]                                                  1440      1132       821        821          0        368          0        764     0        0    2
/dev/shm/PostgreSQL.1436672634                          1024       132       130        130          0          4          0        128     0        0    1
/dev/shm/PostgreSQL.3104938386                           112         4         1          1          0          4          0          0     0        0    1
/dev/zero (deleted)                                   8624208     10352      5070       5070          0       9780          0        572     0        0    1
/usr/lib/postgresql/18/lib/auto_explain.so              20         8         0          0          0          8          0          0     0        0    5
/usr/lib/postgresql/18/lib/pg_stat_statements.so        44         8         0          0          0          8          0          0     0        0    5
/usr/lib/locale/locale-archive                        2980        60        19          0         60          0          0          0     0        0    1
/usr/lib/x86_64-linux-gnu/libffi.so.8.1.2               48         8         0          0          0          8          0          0     0        0    5
/usr/lib/x86_64-linux-gnu/libgpg-error.so.0.33.1       160         8         0          0          0          8          0          0     0        0    5
/usr/lib/x86_64-linux-gnu/libgmp.so.10.4.1             516         8         0          0          0          8          0          0     0        0    5
....
/SYSV00ce5741 (deleted)                                  4         0         0          0          0          0          0          0     0        0    1
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2         208        80        18          9         64          8          0          8     0        0    5
[stack]                                                132        36        27         27          0         12          0         24     0        0    1
[vvar]                                                  16         0         0          0          0          0          0          0     0        0    1
[vdso]                                                   8         4         0          0          4          0          0          0     0        0    1
-------------------------------------------------------------------------------------------------------------------------------------------------------------
Total                                                 8701512     20380      8553       6841       6356      11760        164       2100     0        0  251
```

Let’s Run Some Heavy Query

HOW 2026 Hello Open-source World

credativ

- Let’s run some heavy queries and look into `smaps` again

```
## top command output after query run
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 190747 postgres  20   0 8701848   8.2g   8.1g S   0.0  26.3   0:48.67 postgres: postgres postgres 172.18.0.1(40278) idle

## smaps numbers after query run
Path                                          Size        Rss        Pss  Pss_Dirty  Shr_Clean  Shr_Dirty  Prv_Clean  Prv_Dirty      Swap  SwapPss  Cnt
-------------------------------------------------------------------------------------------------------------------------------------------------------
/usr/lib/postgresql/18/bin/postgres         9296       6508       3255         79       4176        164       2112         56         0        0    5
[anonymous]                                 1708        704        598        598          0        120          0        584         0        0   21
[heap]                                      1776       1516       1208       1208          0        364          0       1152         0        0    2
/dev/shm/                                   1136        148        143        143          0          8          0        140       980        0    2
/dev/zero (deleted)                      8624208    8532008    8443508    8443508          0     143376          0    8388632         0        0    1
/usr/lib/postgresql/18/lib/                   64         44         22          8         28          8          0          8         0        0   10
/usr/lib/locale/locale-archive              2980         68         19          0         68          0          0          0         0        0    1
/usr/lib/x86_64-linux-gnu/                 60520       5020       1376        167       3556       1284        156         24         0        0  205
/SYSV00ce5741 (deleted)                        4          0          0          0          0          0          0          0         0        0    1
[stack]                                      132         44         44         44          0          0          0         44         0        0    1
[vvar]                                        16          0          0          0          0          0          0          0         0        0    1
[vdso]                                         8          4          0          0          4          0          0          0         0        0    1
-------------------------------------------------------------------------------------------------------------------------------------------------------
Total                                    8701848    8546064    8450173    8445755       7832     145324       2268    8390640       980        0  251
```

How Much Memory is Really Used?

HOW 2026 Hello Open-source World

credativ

- I did some playing with multiple sessions with/without parallelism
- Leading to different RSS numbers in the top command -> Let's dive into `smaps` for all these sessions

```
## top command
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
190747 postgres  20   0 8701868   8.1g   8.1g S   0.0  26.2   0:48.69 postgres: postgres postgres 172.18.0.1(40278) idle
206119 postgres  20   0 8702808   4.7g   4.7g S   0.0  15.2   0:21.81 postgres: postgres postgres 172.18.0.1(44010) idle
219065 postgres  20   0 8802384   8.2g   8.1g S   0.0  26.6   2:56.64 postgres: postgres postgres 172.18.0.1(52912) idle
228832 postgres  20   0 8709912   3.4g   3.4g S   0.0  11.1   0:11.10 postgres: postgres postgres 172.18.0.1(60090) idle
230390 postgres  20   0 8701340   8.1g   8.1g S   0.0  26.3   0:51.89 postgres: postgres postgres 172.18.0.1(44802) idle

## smaps summaries with /dev/zero
Path                                           Size         Rss         Pss  Pss_Dirty  Shr_Clean  Shr_Dirty  Prv_Clean  Prv_Dirty       Swap    SwapPss  Cnt
-------------------------------------------------------------------------------------------------------------------------------------------------------------
Total for /proc/190747/smaps                 8701868     8529172     2196188    2195833       2960    8525332          0        880      61784       1116  256
Total for /proc/206119/smaps                 8702808     4928096     1119095    1118712       3120    4924968          0          8      63996       2112  258
Total for /proc/219065/smaps                 8802384     8635640     2296848    2295726       5472    8531892         12      98264      64896       4078  252
Total for /proc/228832/smaps                 8709912     3609444      798269     795595       8660    3590600         60      10124      60936        100  252
Total for /proc/230390/smaps                 8701340     8542712     2202431    2199069       8660    8531564        748       1740      60768         99  251
-------------------------------------------------------------------------------------------------------------------------------------------------------------
                                            43618312    34285064     8611831    8613495      34872   34193356        820      19916     312480      17405 1279
## smaps summaries without /dev/zero
Path                                           Size         Rss         Pss  Pss_Dirty  Shr_Clean  Shr_Dirty  Prv_Clean  Prv_Dirty       Swap    SwapPss  Cnt
-------------------------------------------------------------------------------------------------------------------------------------------------------------
Total for /proc/190747/smaps                   77660        4416        1291        936       2960        576          0        880       3508       1116  255
Total for /proc/206119/smaps                   78600        3708         448         65       3120        580          0          8       5720       2112  257
Total for /proc/219065/smaps                  178176      104316       99427      98305       5472        584         12      98248       6620       4078  251
Total for /proc/228832/smaps                   85704       19420       12853      10179       8660        576         60      10124       2660        100  251
Total for /proc/230390/smaps                   77132       11716        5143       1781       8660        584        748       1724       2492         99  250
-------------------------------------------------------------------------------------------------------------------------------------------------------------
                                              499272      169576      118162     110266      34872       2900        820      19984      18300      17405 1274
```

HOW 2026
Hello Open-source World

credativ

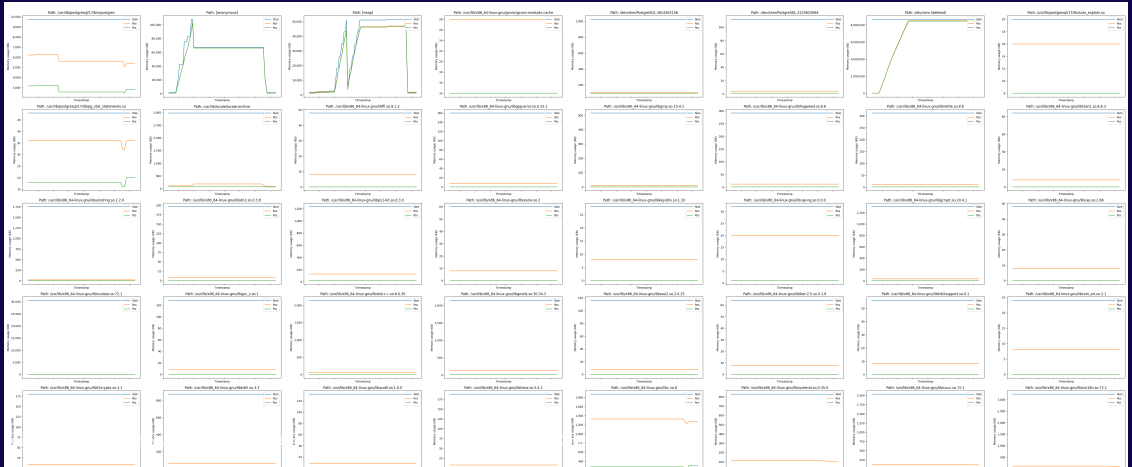
Where is work_mem hidden?

Logging smaps for PostgreSQL process

HOW 2026
Hello Open-source World

credativ

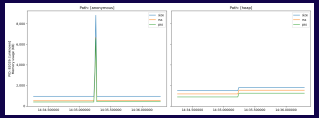
- Scripts scrape smaps data for PostgreSQL connections - each access slows down the process
- Another script parses smaps snapshots and plots a grid (Virt Size, RSS, PSS) for all paths

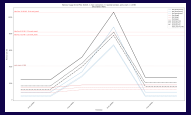


Sorting operation - ORDER BY

- RAM = 32 GB, JIT = off, shared_buffers = ’8GB’, work_mem = ’4MB’

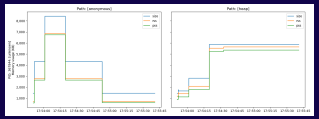
```
SELECT * FROM generate_series(1, 100000) AS t(val) ORDER BY val;
```

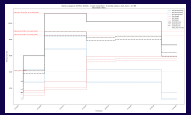




2x work_mem

```
SELECT * FROM generate_series(1, 100000000) AS t(val) ORDER BY val DESC;
```





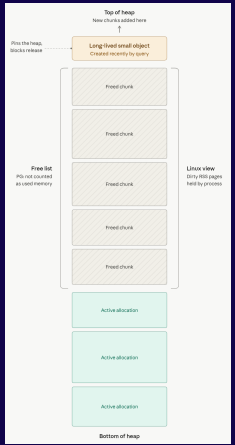
2.2x work_mem

Why Heap memory stays high after query?

HOW 2026
Hello Open-source World

credativ

- Heap memory grows by allocating new chunks at the top of the heap
- Deallocation must be done from the top of the heap downwards
- A query can create a long-living small object at the top
- Deallocated lower chunks are only placed on the free list
- PostgreSQL internally does not count this memory as used
- Linux sees them as dirty memory blocks belonging to the process



HOW 2026
Hello Open-source World

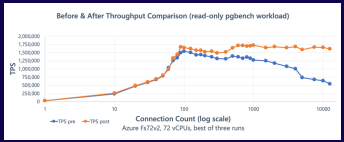
credativ

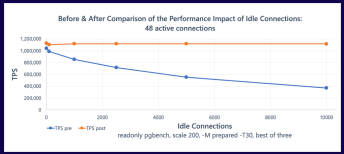
PostgreSQL Internal Limits

MVCC Snapshot Overhead

- MVCC snapshot management scaled very poorly in pre-v14
- Each session has transaction ID horizon / MVCC snapshot
- New transactions needed to check all sessions' snapshots
- Scans of Process Array require ProcArrayLock -> LWLock contention
- Big number of idle sessions degraded throughput of transactions/sec
- v14+ improved this significantly, throughput is now stable
- Each row has "visible since" (xmin) and "visible until" (xmax) XIDs
- Snapshot defines which XIDs are visible to transaction
- More connections -> more XIDs to track -> more overhead

- Improving Postgres Connection Scalability: Snapshots





PostgreSQL Relation Locking Limits

HOW 2026 Hello Open-source World

credativ

- Another bottleneck - too many locked objects in the session
- PG 9.2 - v17: hard coded setting FP_LOCK_SLOTS_PER_BACKEND = 16
  - Capacity of ”fast-path” locking mechanism - relations locks are acquired locally
  - Locks over this capacity require accessing central shared lock table
  - Slows down the system due to contention on the shared lock table
  - Increases latency linearly, planning time grows accordingly
  - Causes huge spikes of ”LWLock” waits in pg_stat_activity
  - Good enough for 2010s query patterns with typically few tables per query
  - Limiting for complex schemas of 2020s with many tables/indexes per query
  - Partitioning made it much worse for some types of workloads
- PG 18 increases this limit to max_locks_per_transaction setting (default 64)
- Planning time and latency seems to be more or less flat, no linear increase

HOW 2026
Hello Open-source World

credativ

Summary

Summary of Key Takeaways (1/2)

HOW 2026 Hello Open-source World

credativ

- **Process Architecture & Memory Overhead**
  - Multi-process model degrades with high connection counts
  - Too many context switches, TLB misses, and heavy memory footprint
  - Interprocess communication relies on Linux `tmpfs`
- **Memory Allocation Management**
  - **Huge Pages** reduce the size of Linux memory mapping tables for large `shared_buffers`
  - Transparent Huge Pages (THP) disabled to prevent reallocation latency
- **NUMA Architecture**
  - Pinning PostgreSQL to specific NUMA nodes mostly useful for workload isolation
  - For mainstream workloads, default Linux NUMA balancing is often sufficient

Summary of Key Takeaways (2/2)

- **CPU Schedulers & Limits**
  - EEVDF scheduler (kernel 6.6+) better handles ”bursty” connections
  - Aim for 2–4 active OLTP connections per CPU core (NVMe/SSD + Huge Pages)
- **Memory Usage & Heap Bloat**
  - Small `work_mem` allocations use heap (`brk()`) -> memory bloat
  - Larger allocations use `mmap()` -> separate anonymous mappings
- **PostgreSQL Internal Bottlenecks Solved**
  - **PG 14+**: Resolved major MVCC snapshot bottlenecks
  - **PG 18+**: Improves fast-path relation locking capacity

HOW 2026
Hello Open-source World

Thank you for your attention!

credativ

Questions?

All my slides

Recorded talks