HOW 2026
Hello Open-source World

credativ

The Alchemy of Shared Buffers

Balancing Concurrency and Performance

Josef Machytka <josef.machytka@credativ.de>

2026-04-26 - 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 IvorySQL community for supporting my participation on HOW2026 conference

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

Topics Covered

- Linux Memory Management
- PostgreSQL Architecture Overview
- Linux Shared Memory
- PostgreSQL Shared Memory
- Huge Pages
- Shared Buffers Internals
- Shared Buffers Usage in Connections
- Shared Buffers and NUMA Architecture

- Versions of software:
  - PostgreSQL 18
  - Linux Kernel 6.12 (Debian 13)
  - x86-64 arch (Intel/AMD)
  - ARM64 arch (Phytium)

- Repository: github.com/josmac69/shared_buffers_workshop

- Pictures without credits created by Claude Opus 4.7 and Gemini 3.1 Pro

HOW 2026
Hello Open-source World

credativ

Linux Memory Management

Memory Management on Linux

HOW 2026 Hello Open-source World

credativ

- Memory must be shared efficiently between processes
- Empty memory is wasted memory
- Linux uses virtual memory to abstract physical memory
- Allows to use more memory than physically available
- Gives each process an illusion of having all memory to itself
- Each process has its own virtual address space

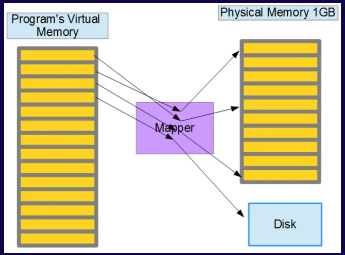
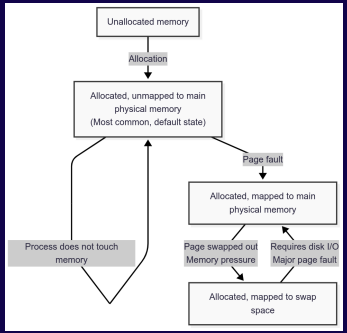


Image from the article
Virtual Memory & Physical Memory

Stages of Memory Allocation

1. Unallocated memory
2. Allocated, unmapped to main physical memory
3. Allocated, mapped to main physical memory
4. Allocated, mapped to swap space



- State 2 is the most common, default state
- If process not really touches memory, it stays in 2
- Transition to state 3 is a page fault
- If transition to 3 requires disk IO -> major page fault
- State 4 = page swapped out due to memory pressure

Memory Management Unit (MMU)

- Hardware unit, translates virtual to physical addr
  - 64-bit x86 - base: 4 KB / huge: 2 MB, 1 GB
  - ARM64 - 4 KB / 2 MB, 1 GB
  - ARM64 - 16 KB / 32 MB
  - ARM64 - 64 KB / 512 MB
- For quick access - Translation Lookaside Buffer (TLB)
- If not in TLB, CPU must ”walk” through page tables (slow)

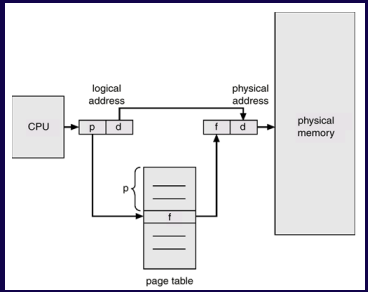


Image from the article
MMU & Virtual Memory

Memory Sizes based on states of allocation

HOW 2026 Hello Open-source World

credativ

1. Unallocated memory
2. Allocated, unmapped
3. Allocated, mapped to main memory
4. Allocated, mapped to swap space

- Virtual memory size - all memory process requested (can be in states 2, 3, 4)
- Resident Set Size (RSS) - memory process is actually using (state 3), including shared libraries and buffers
- Unique Set Size (USS) - memory process is using without shared libraries and buffers

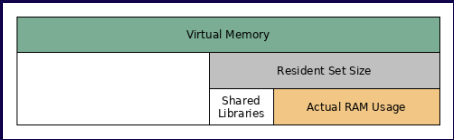
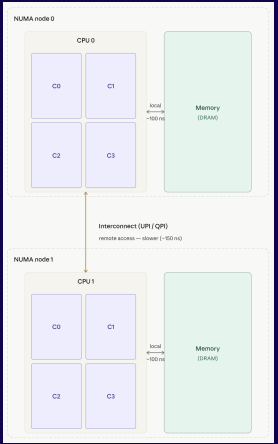


Image from the article Virtual Memory vs. Resident Set Size

None Uniform Memory Access (NUMA)

- NUMA is a computer memory design used in multiprocessing
- It separates architecture into NUMA nodes
- Each node has its own CPUs and memory banks
- It reduces contention for memory access
- Increases memory bandwidth, reduces latency
- PostgreSQL 18 got initial support for NUMA awareness
- Should improve performance on multi-node/multi-socket servers



Huge Pages

- Managing many standard 4 KB small pages causes significant overhead
- Huge pages 2 MB / 1 GB are feature of Memory Management Unit (MMU)
- Reduce the number of page table entries & Translation Lookaside Buffer (TLB) misses
- HP 2 MB = 512 x 4 KB pages, HP 1 GB = 512 x 2 MB / 262144 x 4 KB pages
- Limitations of huge pages:
  - Cannot be swapped out
  - Cannot be moved - pinned in memory
  - Cannot be used by processes which do not support huge pages

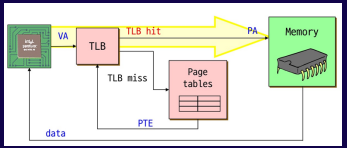


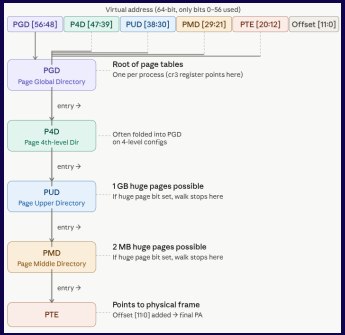
Image from the article Virtual Memory

Linux Memory Allocation Management

HOW 2026 Hello Open-source World

credativ

- Linux implements multiple levels of memory mapping tables (4 or 5)
- Mapping translates 64bit 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 contains pointers to next level table or final mapping
- PUD and PMD have PS bit set when huge pages are used

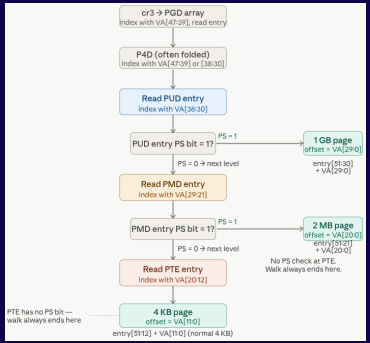


Standard Pages vs Huge Pages

HOW 2026 Hello Open-source World

credativ

- 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
```

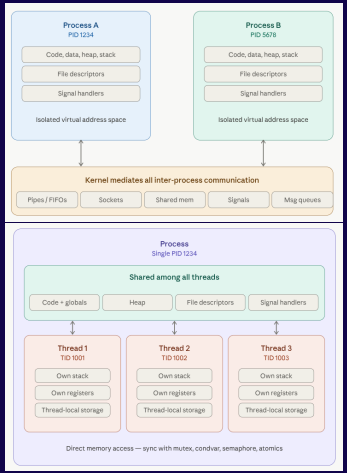
HOW 2026
Hello Open-source World

credativ

PostgreSQL Architecture Overview

Multi-process PostgreSQL Architecture

- 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 have been optimized lately
- Ongoing discussion about switch to threads
  - Threads are now reliable and considered lightweight
  - But handling memory safety depends on developers

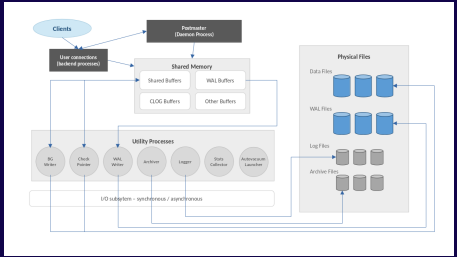


Multi-process PostgreSQL Architecture

HOW 2026 Hello Open-source World

credativ

- **Postmaster**:
  - The main PostgreSQL process
  - Starts all other processes as children
- **Client backends**:
  - One process per connection
  - Execute SQL, make pages dirty, generate WAL records
- **WAL writer**:
  - Flushes WAL buffers to disk in the background
- **Background writer**:
  - Continuously writes dirty shared buffers
  - Smooths out I/O between checkpoints
- **Checkpointer**:
  - Orchestrates writes and fsyncs of data and SLRU buffers
  - Writes checkpoint WAL records
  - Recycles WAL segments



Linux Processes

HOW 2026 Hello Open-source World

credativ

- Typical processes started by PostgreSQL

```
/usr/lib/postgresql/18/bin/postgres -c config_file=/etc/postgresql/18/main/postgresql.conf
postgres: 18/main: logger
postgres: 18/main: checkpointer
postgres: 18/main: background writer
postgres: 18/main: walwriter
postgres: 18/main: autovacuum launcher
postgres: 18/main: logical replication launcher
postgres: 18/main: io worker 0
postgres: 18/main: io worker 1
postgres: 18/main: io worker 2
postgres: 18/main: io worker 3

postgres: 18/main: postgres testdb [local] idle
postgres: 18/main: postgres testdb [local] SELECT
postgres: 18/main: postgres testdb 172.18.0.1(43364) idle
```

Background Writer

HOW 2026 Hello Open-source World

credativ

- ”Best effort” process for cleaning shared buffers
- Connections do not need to perform writes of dirty pages
- Checks how many dirty buffers should be written to disk
- Runs without a transaction, without a database connection
- In case of SIGINT/SIGTERM simply exits, has nothing to cancel
- `bgwriter_flush_after`: flush pending writes after this amount of data pages (64=512KB)
- Does not use fsync(), only sync_file_range() for performance
- But it makes fsync at checkpoint much cheaper

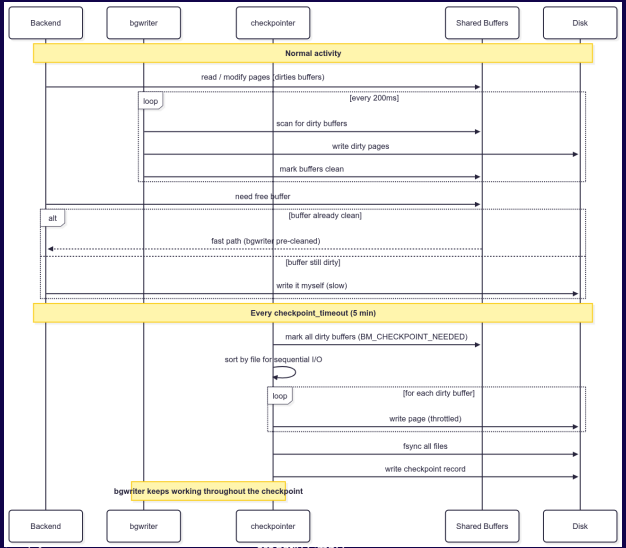
Background Writer vs Checkpointer

HOW 2026 Hello Open-source World

credativ

Characteristic	Background Writer	Checkpointer
Goal	Low-latency backend allocations - connections can find space in memory instantly	High-reliability data persistence
Target Buffer Pool	Only cold (low usage) dirty buffers	All dirty buffers marked for checkpoint
I/O Pattern	Continuous, small bursts of writes	Periodic, large-volume writes
Stats View (v18)	pg_stat_bgwriter	pg_stat_checkpointer (New in 18)
Interaction with WAL	Does not recycle WAL; just flushes data	Marks WAL redo point and enables recycling

Background Writer and Checkpointer



HOW 2026
Hello Open-source World

credativ

Linux Shared Memory

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

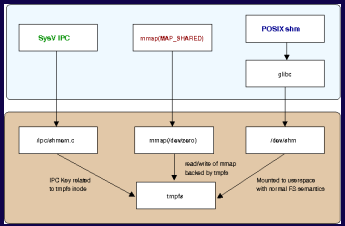


Image from the article Shared memory on Linux

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 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
```

Shared Memory on Linux

HOW 2026 Hello Open-source World

credativ

- Data pages in `tmpfs` are anonymous memory pages
  - -> not backed by any file on disk, contents exist only in RAM
- Can be swapped out if necessary - like other memory pages
  - Kernel's page-replacement algorithm decides when
  - Can be security risk for sensitive data - encrypted swap recommended
  - -> swapping can degrade PostgreSQL performance
- `tmpfs` supports `Transparent Huge Pages` (THP)
  - Improves performance for large memory allocations
  - But can cause performance degradation for PostgreSQL
  - -> PG docs recommend to disable THP for DB servers
- It has also NUMA allocation policy option
  - -> local on current CPU / bind to specific NUMA node
- Oversizing tmpfs & swap disabled can deadlock the system
  - -> if OOM killer is disabled it cannot free memory

Mount options of tmpfs

HOW 2026 Hello Open-source World

credativ

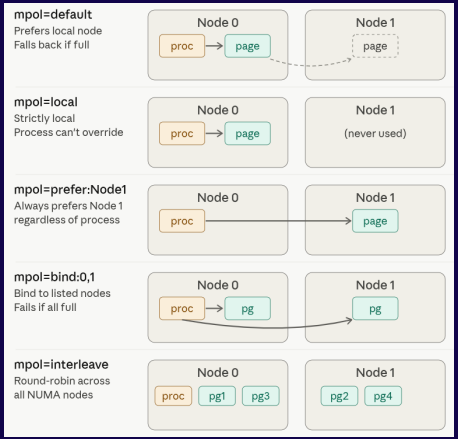
- Current settings can be checked with `findmnt -t tmpfs`

```
TARGET                                                  SOURCE FSTYPE OPTIONS
/dev/shm                                                 tmpfs  tmpfs  rw,nosuid,nodev,inode64
/run                                                     tmpfs  tmpfs  rw,nosuid,nodev,noexec,relatime,size=3252812k,mode=755,inode64
/run/lock                                              tmpfs  tmpfs  rw,nosuid,nodev,noexec,relatime,size=5120k,inode64
/run/credentials/systemd-journald.service              tmpfs  tmpfs  ro,nosuid,nodev,noexec,relatime,nosymfollow,size=1024k,nr_inodes=1024,mode=700,inode64,noswap
/run/credentials/systemd-resolved.service              tmpfs  tmpfs  ro,nosuid,nodev,noexec,relatime,nosymfollow,size=1024k,nr_inodes=1024,mode=700,inode64,noswap
/run/credentials/systemd-cryptsetup@nvme0n1p3_crypt.service tmpfs  tmpfs  ro,nosuid,nodev,noexec,relatime,nosymfollow,size=1024k,nr_inodes=1024,mode=700,inode64,noswap
/run/user/1000                                         tmpfs  tmpfs  rw,nosuid,nodev,relatime,size=3252808k,nr_inodes=813202,mode=700,uid=1000,gid=1000,inode64
/tmp                                                     tmpfs  tmpfs  rw,nosuid,nodev,size=16264048k,nr_inodes=1048576,inode64
```

- On Debian 13 options are stored in `/etc/mtab`
  - `size` - maximum size of tmpfs, default usually 50% of RAM
  - `nr_inodes` - maximum number of inodes, default half of total physical RAM pages
  - `noswap` - disables swapping of tmpfs files
  - `mpol` - sets NUMA policy - allows bind to specific NUMA node

NUMA policy of tmpfs

- `mpol=default`
  - RAM is allocated on the same NUMA node as the process
  - Process can call `set_mempolicy` to change it
  - If node is full, kernel will allocate on closest node
- `mpol=local`
  - RAM is allocated on the same NUMA node as the process
  - Attempt of process to change policy is ignored
- `mpol=prefer:Node`
  - Kernel will always try to allocate on specified NUMA node
  - Regardless of current process location
  - If node runs out of memory, it will allocate on other nodes
- `mpol=bind:NodeList`
  - Binds to specific NUMA nodes - strict policy
  - If nodes run out of memory, allocation will fail
- `mpol=interleave`
  - interleave pages across NUMA nodes



HOW 2026
Hello Open-source World

credativ

PostgreSQL Shared Memory

History of PostgreSQL Shared Memory

HOW 2026 Hello Open-source World

credativ

- Originally PostgreSQL used `System V` (SysV) for Inter-Process Communication (IPC)
  - Usage of SysV shared memory segment was later discouraged
  - Its default size was limited by default to 32MB
  - Higher sizes required reconfiguration of OS kernel parameters
  - See in Absurd Shared Memory Limits blog post by Robert Haas (2012)
  - Documentation still shows how to configure SysV shared memory
  - SysV allowed PG to detect multiple postmasters accessing the same data directory
  - Now this is done via `postmaster.pid` file -> empty PID file can prevent start (!)

- Since PG 9.3, default is POSIX shared memory on Linux
- Parameter `shared_memory_type` controls the type of shared memory
  - `mmap` - for anonymous shared memory allocated using mmap (default on Linux)
  - `sysv` - for System V shared memory allocated via shmget
  - `windows` - for Windows named shared memory

PostgreSQL Shared Memory

HOW 2026 Hello Open-source World

credativ

- Main shared memory area allocated on server startup by the `postmaster`
- Postmaster sits in loop waiting for connections -> woken by kernel -> accepts connection
- Creates new process for a connection using `fork()` system call
  - Child process inherits parent's memory mapping - mapped as `MAP_SHARED`
  - All processes see the same shared memory region
  - Changes in memory are visible to all processes

- PostgreSQL can use `Huge Pages` for shared buffers and some other shared memory objects
  - Reduce TLB (Translation Lookaside Buffer) misses
  - Hence reduce CPU usage = improve performance

Huge Pages in PostgreSQL

HOW 2026 Hello Open-source World

credativ

- Usage of Huge Pages is not entirely straightforward
  - By default Huge Pages are not enabled on Linux -> `vm.nr_hugepages = 0`
  - Check `sysctl vm.nr_hugepages` to see the current setting
  - -> PG setting `huge_pages = try` does not have any effect

- Wrongly configured `Huge Pages` can cause memory issues
  - Configured number of Huge Pages is pre-allocated at Linux boot time
  - Pages are pinned in memory -> not swappable, guaranteed TLB efficiency
  - Only processes which implement usage of Huge Pages can use this area
  - It is not available for other processes
  - Configured number can be dynamically changed at runtime
  - But only downsizing is recommended

Huge Pages in PostgreSQL

HOW 2026 Hello Open-source World

credativ

- `cat /proc/meminfo |grep -i hugepagesize` -> see Huge Page size
- Typical Huge Page size is 2 MB, but 1 GB is also possible
- -> read-only parameter `shared_memory_size_in_huge_pages`
- -> shows how much huge pages would be required
- -> can be checked before starting PostgreSQL -
  `postgres -D $PGDATA -C shared_memory_size_in_huge_pages`
- Example: 8GB shared buffers -> 4096 Huge Pages 2048kB big
- But check of `shared_memory_size_in_huge_pages` shows always bigger number
- -> includes other shared memory objects as well
- Even PG docs recommend to configure on Linux even more than this value
- ... (continuation on the next slide)

Huge Pages in PostgreSQL

HOW 2026 Hello Open-source World

credativ

- Set PostgreSQL to use only Huge Pages -> `huge_pages = on`
- Start PostgreSQL -> if it fails, start it again in DEBUG mode
- If it starts, check `shared_memory_size_in_huge_pages` - how many Huge Pages are used
- Check usage of Huge Pages on Linux -> `cat /proc/meminfo |grep -i hugepages`
  - HugePages_Total: 1024
  - HugePages_Free: 800
  - HugePages_Rsvd: 50 (promised pages)
  - -> number of really used Huge Pages: `HugePages_Total` - `HugePages_Free` = 224
- Shrink `vm.nr_hugepages` to the number of Huge Pages used by PostgreSQL
- -> Number seems to be stable for given PostgreSQL and Linux versions
- -> Performance gain seems to be around 10-15% on longer queries

- Talk: PostgreSQL and Hugepages
- Cybertec: Huge Pages and PostgreSQL

Huge Pages in PostgreSQL

HOW 2026 Hello Open-source World

credativ

- Performance improvements around 10-15% on longer queries
- Memory allocation tables significantly smaller
  - 4 KB pages: 1000 connections, 8 GB buffers = 16 GB PTE tables (!!!)
  - 2MB huge pages: 1000 connections, 8 GB buffers = 32 MB PMD tables
- Better cache utilization of Translation Lookaside Buffer (TLB)
  - Fewer TLB misses, cheaper CPU context switching

PostgreSQL And Transparent Huge Pages

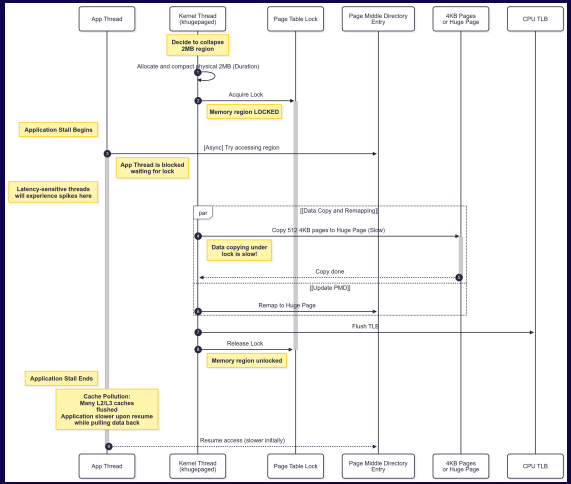
HOW 2026 Hello Open-source World

credativ

- Distros enable dynamic `Transparent Huge Pages (THP)` by default
  - Introduced to ”democratize” Huge Pages benefits for all applications
  - Can be swapped out -> kernel breaks them back into 4kB pages
  - `khugepaged` - scans memory for contiguous memory regions
  - `kcompactd` - compacts memory / copies pages (on NUMA one per node)
  - `kswapd` - swaps memory
  - Merging or splitting causes latency spikes and locks on memory pages
  - -> THP are not recommended for PostgreSQL due to performance issues

- Why THP are not good for PostgreSQL?
  - During run of query connection allocates memory for processing
  - In smaps marked as ”[anonymous]” and ”[heap]”
  - This allocation is more or less work_mem size -> few MB or dozens of MB
  - -> attempts to merge these pages cause latency spikes and locks on memory pages

PostgreSQL And Transparent Huge Pages



Huge Pages / THP And Other Databases

HOW 2026 Hello Open-source World

credativ

- Oracle:
  - Strongly recommends to use Huge Pages
  - -> Uses massive shared memory ”System Global Area” (SGA)
  - Requires disable of THP to avoid memory allocation delays
  - -> Can even break HA features due to delayed heartbeat responses
- MySQL / MariaDB:
  - Supports ”Large Pages” for InnoDB buffer pool, but configuration is more complex
  - Requires disable of THP especially if ”jemalloc” is used
  - (High performance memory allocator)
- MongoDB 8.0+ -> HP not supported, enable THP (v7.0 or earlier - disable THP)
- Redis -> HP not supported (madvise), disable THP - big latency spikes can occur
- Couchbase, Aerospike, ClickHouse -> HP not supported, disable THP

PostgreSQL SysV Interlock Segment

HOW 2026 Hello Open-source World

credativ

- PostgreSQL implements ”separation of concerns” principle
- It still allocates a tiny `SysV interlock segment` on startup
- Used to ”advertise” the presence of a running PostgreSQL instance
  - Size is small - typically a few kilobytes
  - It holds `PGShmemHeader` structure - used for inter-process synchronization
  - Defined in `src/include/storage/pg_shmem.h`
  - Contains cluster identity and other control metadata
  - All PG processes attach to it for synchronization
  - Allows instance discovery and avoiding split-brain conflict
  - Used for detection of multiple postmasters

```
## command ipcs shows SysV IPC objects
postgres=# ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00d295be 0          postgres   600        56         16
```

Other Shared Memory Objects

- `WAL buffers` - caches WAL records before writing to disk
- `SLRU buffers` - CLOG, MultiXact, CommitTS etc. control structures
- `Lock table` - tracks locks held by transactions
- `Other objects` - various control structures
- Size depends on settings: `max_connections`, `max_locks_per_transaction`
- The `shared_memory_size` parameter reports the size of the main shared memory area (MB)
- Can be exhausted during some operations -> ”out of shared memory” error
- Change in settings for shared memory requires restart

```
select name, setting, unit from pg_settings where name like '%shared%' order by name;

               name                | setting | unit
-----------------------------------+---------+------
 dynamic_shared_memory_type        | posix   |
 min_dynamic_shared_memory         | 0       | MB
 shared_buffers                    | 16384   | 8kB
 shared_memory_size                | 179     | MB
 shared_memory_size_in_huge_pages  | 90      |
 shared_memory_type                | mmap    |
 shared_preload_libraries          |         |
```

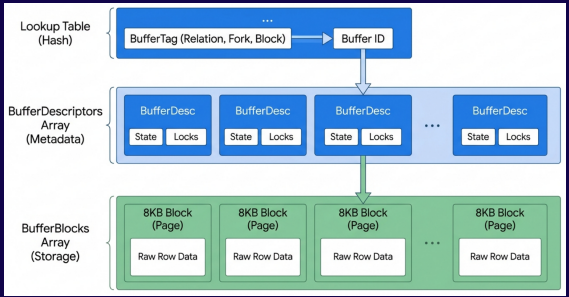
HOW 2026
Hello Open-source World

credativ

Shared Buffers Internals

Shared Buffers

- The biggest & most discussed PostgreSQL memory object
  - In-memory copy of tables and indexes data blocks
  - Allocated on server startup, Shared among all connections
  - Blocks kept / evicted based on frequency of access
  - At the beginning exist only as virtual memory



Shared Buffers

HOW 2026 Hello Open-source World

credativ

- `EXPLAIN ANALYZE` shows how successful was query in using shared buffers
  - `Buffers: shared hit=X, read=Y, dirtied=Z, written=W`
  - shared = blocks found in shared buffers
  - read = blocks not found in shared buffers
  - dirtied = blocks modified in shared buffers
  - written = blocks written to disk

- `pg_stat_activity` - `wait_event_type / wait_event`
  - `BufferPin` - waiting for exclusive access to a data buffer
  - -> BufferPin
  - `LWLock` - light weight lock is held
  - -> BufferContent - Waiting to access a data page in memory (hot pages)
  - -> BufferMapping - Waiting to associate a data block with a buffer (high eviction rate)

Shared Buffers Max Size

HOW 2026 Hello Open-source World

credativ

- Size is internally stored as number of data pages
  - In source code in `src/backend/utils/init/globals.c`
  - ”int NBuffers = 16384;” (= 128MB)
  - NBuffers limit is theoreticalyy INT_MAX = 2,147,483,647 -> 16 TB
  - -> but capped in code to INT_MAX /2 = 1,073,741,823 -> 8 TB
  - -> capped in `src/backend/utils/misc/guc_tables.c` - line 2369

- But this size would be very challenging
  - Just Descriptors would take 64 GB
  - Massive TLB trashing (Huge Pages would be required)
  - Most likely extended checkpoints times
  - Non-linear scanning costs in the buffers eviction algorithm

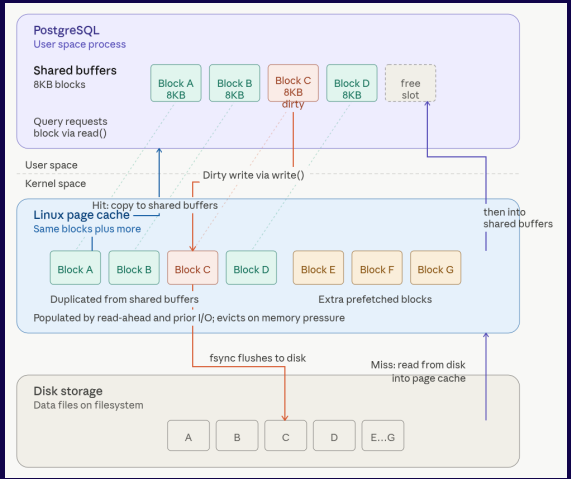
Shared Buffers Recommended Size

HOW 2026 Hello Open-source World

credativ

- Recommended `25% of the available memory` -> but why? And is it still true?
  - Recommendation is quite old, when machines had only few GB of RAM
  - Idea was to keep as much data in Linux page cache as possible
  - And avoid big double caching (PG shared buffers + Linux page cache)
- New PG 18 Asyc IO reads quicker from cache with multiple workers
- When we can benefit from larger shared buffers
  - Workload repeatedly touches the same data (high reuse factor)
  - -> Content of shared buffers must be relatively static
  - Reused set of pages fits in shared buffers
  - -> Frequent big analytical queries are the good candidates here
  - Additional memory would not be beneficial elsewhere
  - PostgreSQL policies will even allow to use bigger shared buffers fully
  - -> BAS_BULKREAD, BAS_BULKWRITE, BAS_VACUUM

Double Caching



Use Cases for Larger Buffers

HOW 2026 Hello Open-source World

credativ

- Hot-tier time-series workloads
  - Heavy selects over last N hours/days
  - Hot partitions are small, reused intensely
- Gaming states, real-time pricing
  - Small datasets, extreme read and update frequences
  - Cache misses are unacceptable
- Database fully fits into shared buffers
  - Many databases are just dozens of GBs, servers can be bigger
  - Either large shared_buffers and small Linux page cache
  - Or small shared_buffers and large Linux page cache
- Tuning shared_buffers for OLTP and data warehouse workloads

Possible Problematic Factors

HOW 2026 Hello Open-source World

credativ

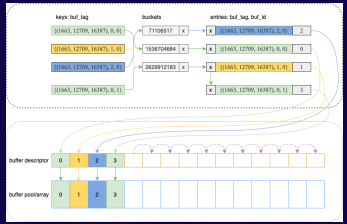
- PostgreSQL uses MVCC - new version of tuple is created on update/delete
  - Dead tuples remain in shared buffers
  - Take up space of active tuples
  - -> Vacuum needs to remove them to free up space
- Vacuum cannot reclaim space in data pages in cases:
  - Long running transactions block TIDs and prevent cleanup
  - Abandoned prepared transactions -> pg_prepared_xacts
  - Replication slot is not consumed - holds back catalog_xmin
  - -> replication slots requires catalog consistency
  - hot_standby_feedback is enabled + standbys run long queries

Shared Buffers Structure

HOW 2026 Hello Open-source World

credativ

- `BufferTag` structure
  - Identifies which disk block is in which buffer
  - Tablespace OID, database OID, Relfile num, Fork num, Block num
  - Fork = file: 0=main, 1=free space map, 2=visibility map, 3=unlogged
- `Buffer Descriptors`
  - metadata for each buffer block - locks, dirty flag, usage count, tags
  - max 64 bytes per descriptor -> NBuffers * 64 bytes
- `BufferDescPadded`
  - Padding to 64 bytes of CPU cache line size
  - Alignment for highly concurrent workloads
  - Avoids false sharing, i.e. unintentional cache invalidation
- `BufferLookupEnt` - hash table for fast lookup
  - Contains BufferTag and associated buffer ID
- `BufferBlocks` array
  - Actual data blocks of tables and indexes -> NBuffers * 8kB



(Image from the article
The Amazing Buffer Tag in PostgreSQL)

Shared Buffers Structure

HOW 2026 Hello Open-source World

credativ

```
typedef struct buftag
{
    Oid         spcOid;         /* tablespace oid */
    Oid         dbOid;          /* database oid */
    RelFileNumber relNumber;    /* relation file number */
    ForkNumber  forkNum;        /* fork number */
    BlockNumber blockNum;       /* blknum relative to begin of reln */
} BufferTag;


typedef struct
{
    BufferTag   key;            /* Tag of a disk page */
    int         id;             /* Associated buffer ID */
} BufferLookupEnt;


typedef struct BufferDesc
{
    BufferTag   tag;            /* ID of page contained in buffer */
    int         buf_id;         /* buffer's index number (from 0) */
    /* state of the tag, containing flags, refcount and usagecount */
    pg_atomic_uint32 state;
    int         wait_backend_pgprocno;  /* backend of pin-count waiter */
    int         freeNext;       /* link in freelist chain */
    PgAioWaitRef io_wref;       /* set iff AIO is in progress */
    LWLock      content_lock;   /* to lock access to buffer contents */
} BufferDesc;


#define BUFFERDESC_PAD_TO_SIZE  (SIZEOF_VOID_P == 8 ? 64 : 1)
typedef union BufferDescPadded
{
    BufferDesc  bufferdesc;
    char        pad[BUFFERDESC_PAD_TO_SIZE];
} BufferDescPadded;
```

Operational Logic - Cache Hit

- Tag Initialization -> BufferTag constructed for desired block
- Hash Computation -> hash code calculated from tag
- Shared Lock Acquisition -> partition lock acquired
  - Shared lock allows unlimited concurrent readers
- Lookup -> hash table is searched
- Pinning -> backend retrieves buffer ID from hash table
  - Increments refcount (18 bits) in buffer descriptor (=pins the buffer)
  - Increments usage_count (max value 5)
  - Pin must occur while holding partition lock
  - Without partition lock, buffer could be evicted
- Lock Release -> partition lock released immediately after pinning
  - Buffer is now protected by refcount
  - Cannot be evicted until backend unpins it (decrements refcount by 1, keeps usage_count)

Operational Logic - Cache Miss Path

HOW 2026 Hello Open-source World

credativ

- We assume all buffers are already filled with data
- Tag Initialization -> BufferTag constructed for desired block
- Hash Computation -> hash code calculated from tag
- Shared Lock Acquisition -> partition lock acquired
- Initial Lookup -> hash table is searched - buffer NOT found
- Lock Release -> initial shared lock released
- Victim Selection -> find in Descriptors page with refcount == 0 and usage_count == 0
- Exclusive Lock Acquisition -> exclusive lock for the new page's partition
- Re-Check -> check if page is still not in the buffer
  - Other backend might already successfully loaded the page
  - If so, exclusive lock released, victim returned to Free list
  - If not, New BufferTag mapped to BufferID of victim - added to hash table
- Lock Released -> exclusive lock released
- IO Initialization -> IO is initialized

Clock Sweeping & Free List

HOW 2026 Hello Open-source World

credativ

- Clock Sweeping is used to find pages to evict
  - Only page with refcount == 0 can be evicted
  - If usage_count > 0, it was recently repeatedly accessed
  - Clock sweep will decrease it -> usage_count - 1
  - Page with refcount == 0 and usage_count == 0 -> chosen as victim
  - Actions immediately started for reuse of this buffer

- Free List -> list of pages that are not pinned or are empty and can be used
  - Stored in special structure `BufferStrategyControl`
- Victim -> page found for overwriting because Free list was empty
  - Returned page -> page returned to the Free list due to race condition

Buffer Descriptor State Content

- `pg_atomic_uint32 state` in BufferDesc contains:
  - `Reference Count - refcount` - The lower bits 0-17
  - -> maximum number of concurrent backends that can pin a single buffer
  - -> 18 bits - the maximum value is $2^{18} - 1 = 262,143$
  - -> But in reality it is capped to value of MaxBackends
  - `Usage Count - usage_count` - The bits immediately following the refcount - bits 18-21
  - -> capped to value of 5 - 0=cold, 1-2=warm 3-5=hot
  - `Buffer Flags` - The higher bits - bits 22-31

```
MaxBackends = MaxConnections + autovacuum_worker_slots +
    max_worker_processes + max_wal_senders + NUM_SPECIAL_WORKER_PROCS;

if (MaxBackends > MAX_BACKENDS)
    ereport(ERROR,
            (errcode(ERRCODE_INVALID_PARAMETER_VALUE),
             errmsg("too many server processes configured"), ...
```

Buffer Descriptor State Buffer Flags

HOW 2026 Hello Open-source World

credativ

```
#define BM_LOCKED               (1U << 22)  /* buffer header is locked */
#define BM_DIRTY                (1U << 23)  /* data needs writing */
#define BM_VALID                (1U << 24)  /* data is valid */
#define BM_TAG_VALID            (1U << 25)  /* tag is assigned */
#define BM_IO_IN_PROGRESS       (1U << 26)  /* read or write in progress */
#define BM_IO_ERROR             (1U << 27)  /* previous I/O failed */
#define BM_JUST_DIRTIED         (1U << 28)  /* dirtied since write started */
#define BM_PIN_COUNT_WAITER     (1U << 29)  /* have waiter for sole pin */
#define BM_CHECKPOINT_NEEDED    (1U << 30)  /* must write for checkpoint */
#define BM_PERMANENT            (1U << 31)  /* permanent buffer (not unlogged, or init fork) */
```

Buffer Descriptor State Buffer Flags - Details

HOW 2026 Hello Open-source World

credativ

Buffer State Flag	Description	Operational Significance
`BM_VALID`	The buffer contains a valid data page.	Indicates the page is ready for reading by backends.
`BM_DIRTY`	The page has been modified but not written to disk.	Target for flushing by bgwriter or checkpointer.
`BM_IO_IN_PROGRESS`	An I/O operation (read or write) is currently occurring.	Prevents concurrent I/O on the same block.
`BM_CHECKPOINT_NEEDED`	The page must be written during the current checkpoint.	Coordinates bulk flushing for persistence.
`BM_PERMANENT`	Indicates the page belongs to a logged relation.	Requires WAL logging before flushing.

Pin Leak Danger

HOW 2026 Hello Open-source World

credativ

- `refcount` is managed ”manually” (call of Pin/Unpin)
  - -> a bug in the PostgreSQL kernel or an extension can fail to call UnpinBuffer
  - -> such a buffer can never be evicted from shared buffers
  - -> database can run out of shared buffers and stop
- PostgreSQL uses `ResourceOwner` mechanisms to track pins per-transaction
  - -> ResourceOwner cleanup routine forces the unpin
  - -> If a transaction aborts or commits without unpinning
  - -> balance is restored

PrivateRefCount

HOW 2026 Hello Open-source World

credativ

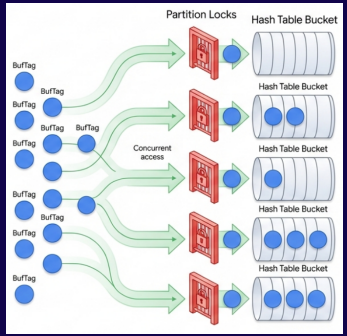
- Process might need to pin the same buffer multiple times in query
  - -> rare, but possible - nested loop joins / index scans / cursor ops
  - Would lead to lock contention in shared buffers and slow down the query
  - -> connections have PrivateRefCountArray - 8 entries, fits to CPU cache
  - Count is not protected by the buffer lock
- Shared buffer lock is acquired when the PrivateRefCount for a buffer is 0
  - -> either when connection needs block for a first time
  - -> or when refcount was decremented back to 0
  - All other pins / unpins are done in user space
- Mechanism is not simple - for our purposes it is enough to know that it exists

Partitioned Buffer Locks

HOW 2026 Hello Open-source World

credativ

- Buffer lookup table is divided into 128 partitions
- Each partition has its own lightweight lock
  - No need to lock the whole table
- Lookup or insertion needs to lock only one partition
- Large buffers and many CPUs can lead to lock contention
  - Implicit constraint on scaling
  - Some forks increase the number of partitions
  - PostgresPro sets NUM_BUFFER_PARTITIONS to 512
  - OrioleDB tries to combine in-memory and on-disk buffers



Partitioned Buffer Locks

HOW 2026
Hello Open-source World

credativ

- Must be a power of 2 - for efficient bitwise arithmetics
- Value 128 is not arbitrary, but calibrated for trade-off
  - Probability of collision for two random tags is 1/128 = 0.78%
  - But high number would cause system stalls for some operations
- DROP TABLE, TRUNCATE TABLE, CHECKPOINT might need to lock all part
  - Overhead of acquiring too many locks sequentially

- Some experiments suggest improvement with higher values
  - On machines with hundreds of CPUs and 256+ GB of RAM
  - Often combined with bigger data blocks -> but this is uncharted territory

Buffer Access Strategies

HOW 2026 Hello Open-source World

credativ

- Clock Sweep algorithm expects Zipfian distribution
- -> relatively small working set is accessed frequently
- Sequential scans are a pathological case
- -> huge scans could replace all pages in the buffer
- Solution is the Buffer Access Strategy (BAS) for bulk operations
- `BAS_BULKREAD` uses a ring buffer of 32 pages (256KB)
- -> can be bigger:
  ring_size_kb += (BLCKSZ / 1024) * io_combine_limit * effective_io_concurrency;
- `BAS_BULKWRITE` uses a ring buffer of 2048 pages (16MB)
- -> to allow more dirty pages to accumulate before flushing
- `BAS_VACUUM` uses a ring buffer - PG16+ vacuum_buffer_usage_limit, default 2MB, 0 = no limit
- -> vacuum is a sequential scan
- Any ring cannot exceed `NBuffers / 8`

HOW 2026
Hello Open-source World

credativ

Shared Buffers Usage in Connections

Shared Buffers In PostgreSQL Connection

HOW 2026 Hello Open-source World

credativ

- Let's look into `/proc/PID/smaps` to see how shared buffers are used in session
- 32GB RAM, PG 18, shared_buffers 8GB, effective_cache_size 24GB, work_mem 64MB
- Detailed view showed 42 different /usr/lib/x86_64-linux-gnu/ libraries
- Find more in my talk PostgreSQL Connection Memory Usage

```
## 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

## 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
....
/SYSV00ce5741 (deleted)                                    4       0       0          0          0          0          0          0     0        0    1
[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
```

Why ”/dev/zero (deleted)” ?

HOW 2026 Hello Open-source World

credativ

- Why shared buffers are mapped as `/dev/zero (deleted)` ” in `smaps` output?
  - PostgreSQL requests ”shared anonymous memory” from OS
  - Using `mmap()` system call with `MAP_ANONYMOUS | MAP_SHARED` flags
  - I.e. ”shared memory with anonymous mapping (not backed by any file)” is requested
  - Described in PG code as ”anonymous mmap()ed shared memory segment”
  - Hence PG setting shared_memory_type = ’mmap’
- How Linux implements this internally?
  - Linux kernel internally instantiates a synthetic file object within `tmpfs`
  - Kernel source code names it ”dev/zero” - legacy from older implementations
  - This internal backing file server only as a handle for memory management
  - It is not linked to Virtual File System (VFS) directory tree
  - Therefore it shows up as ”(deleted)” in `smaps` output
- Hence shared buffers are mapped as `/dev/zero (deleted)`

Let’s Run Some Heavy Query

HOW 2026 Hello Open-source World

credativ

- Let’s run some heavy aggregations over the table not fitting into memory
- Memory 32 GB, table 38 GB, shared_buffers 8 GB, work_mem 64 MB, max_parallel_workers_per_gather = 0

```
## 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
```

Shared Buffers Stay Mapped

HOW 2026 Hello Open-source World

credativ

- I did some playing with multiple sessions with/without parallelism

```
## 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

Inspection and Monitoring

Shared Memory Allocations

HOW 2026 Hello Open-source World

credativ

```
SELECT * FROM pg_shmem_allocations WHERE size > 0 ORDER BY lower(name);
```

name	offset	size	allocated_size
AioBackend	43982891008	122672	122752
AioCtl	43982890880	56	128
AioHandle	43983013760	3446784	3446784
AioHandleData	43992588160	3063808	3063808
AioHandleIOV	43986460544	6127616	6127616
AioWorkerControl	43995652352	520	640
AioWorkerSubmissionQueue	43995651968	272	384
<anonymous>		286502912	286502912
...			
Buffer Blocks	386889344	42949677056	42949677056
Buffer Descriptors	51345024	335544320	335544320
Buffer IO Condition Variables	43336566400	83886080	83886080
Buffer Strategy Status	43802405376	28	128
Checkpoint BufferIds	43420452480	104857600	104857600
Checkpointer Data	43814449152	167772216	167772288
....			
Wal Summarizer Ctl	43982234496	48	128
XLOG Ctl	86528	16803400	16803456
XLogPrefetchStats	16890368	72	128
XLOG Recovery Ctl	16890496	104	128

Basic stats of shared buffers

HOW 2026 Hello Open-source World

credativ

```
select * from pg_buffercache_summary();

 buffers_used | buffers_unused | buffers_dirty | buffers_pinned |  usagecount_avg
--------------+----------------+---------------+----------------+-------------------
          485 |        1965595 |             2 |              0 | 3.350515463917526
(1   )


select * from pg_buffercache_usage_counts();

 usage_count | buffers | dirty | pinned
-------------+---------+-------+--------
           0 | 1965593 |     0 |      0
           1 |     158 |     0 |      0
           2 |      44 |     3 |      0
           3 |      13 |     0 |      0
           4 |      12 |     0 |      0
           5 |     260 |     1 |      0
(6   )
```

Relations in Shared Buffers

HOW 2026 Hello Open-source World

credativ

```
SELECT c.relname AS relation_name,
       c.relkind AS relation_type,
       count(*) AS buffers_in_cache,
       pg_size_pretty(count(*) * (current_setting('block_size')::integer)) AS memory_consumed
FROM pg_buffercache b
JOIN pg_class c ON b.relfilenode = c.relfilenode
    AND b.reldatabase IN (0, (SELECT oid FROM pg_database WHERE datname = current_database()))
GROUP BY c.relname, c.relkind
ORDER BY buffers_in_cache DESC;
```

relation_name	relation_type	buffers_in_cache	memory_consumed
large_random_data	r	5193125	40 GB
numa_chain	r	49388	386 MB
pg_operator	r	27	216 kB
pg_statistic	r	20	160 kB
pg_operator_oprname_l_r_n_index	i	9	72 kB
pg_index	r	6	48 kB

....

Usagecount

HOW 2026 Hello Open-source World

credativ

```
SELECT usagecount,
       count(*) AS buffers,
       pg_size_pretty(count(*) * (current_setting('block_size')::integer)) AS size,
       round(count(*) * 100.0 / (SELECT count(*) FROM pg_buffercache), 2) AS pct_of_cache
FROM pg_buffercache
GROUP BY usagecount
ORDER BY usagecount NULLS FIRST;

 usagecount | buffers |  size   | pct_of_cache
------------+---------+---------+--------------
          0 | 1441775 | 11 GB   |        27.50
          1 | 3800965 | 29 GB   |        72.50
          2 |      28 | 224 kB  |         0.00
          3 |      11 | 88 kB   |         0.00
          4 |      15 | 120 kB  |         0.00
          5 |      86 | 688 kB  |         0.00
```

Dirty Pages

HOW 2026 Hello Open-source World

credativ

```
SELECT isdirty,
       count(*) AS buffers,
       pg_size_pretty(count(*) * (current_setting('block_size')::integer)) AS total_size,
       round(count(*) * 100.0 / (SELECT count(*) FROM pg_buffercache), 2) AS pct_of_cache
FROM pg_buffercache
GROUP BY isdirty;

 isdirty | buffers | total_size | pct_of_cache
---------+---------+------------+--------------
 f       | 5242875 | 40 GB      |       100.00
 t       |       5 | 40 kB      |         0.00
```

HOW 2026
Hello Open-source World

credativ

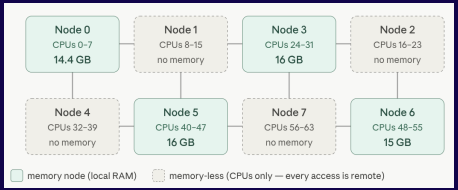
Shared Buffers and NUMA Architecture

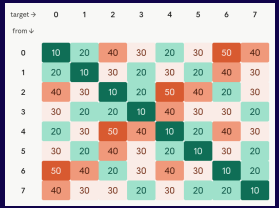
PostgreSQL on ARM64 Phytium

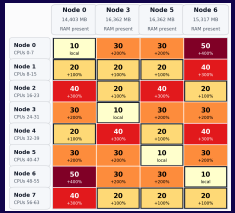
HOW 2026
Hello Open-source World

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





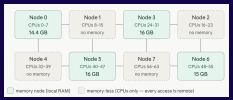


Pinning PostgreSQL on NUMA nodes

HOW 2026
Hello Open-source World

credativ

- Forked processes inherit postmaster's bind policy - memory / CPUs
- Pinning both memory and CPUs to NUMA specific 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

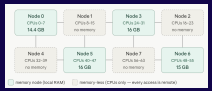


Pinning PostgreSQL on NUMA nodes

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, distance table exaggerates

CPUs per NUMA nodes

HOW 2026 Hello Open-source World

credativ

```
[root@host5 ~]# lscpu |grep -i numa

NUMA                              8
NUMA  0  CPU                      0-7
NUMA  1  CPU                      8-15
NUMA  2  CPU                      16-23
NUMA  3  CPU                      24-31
NUMA  4  CPU                      32-39
NUMA  5  CPU                      40-47
NUMA  6  CPU                      48-55
NUMA  7  CPU                      56-63
```

NUMA nodes HW overview

HOW 2026 Hello Open-source World

credativ

```
[root@host5 node]# numactl --hardware
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 14403 MB
node 0 free: 1271 MB
node 1 cpus: 8 9 10 11 12 13 14 15
node 1 size: 0 MB
node 1 free: 0 MB
node 2 cpus: 16 17 18 19 20 21 22 23
node 2 size: 0 MB
node 2 free: 0 MB
node 3 cpus: 24 25 26 27 28 29 30 31
node 3 size: 16362 MB
node 3 free: 1036 MB
node 4 cpus: 32 33 34 35 36 37 38 39
node 4 size: 0 MB
node 4 free: 0 MB
node 5 cpus: 40 41 42 43 44 45 46 47
node 5 size: 16362 MB
node 5 free: 1084 MB
node 6 cpus: 48 49 50 51 52 53 54 55
node 6 size: 15317 MB
node 6 free: 953 MB
node 7 cpus: 56 57 58 59 60 61 62 63
node 7 size: 0 MB
node 7 free: 0 MB
```

NUMA nodes HW overview

HOW 2026 Hello Open-source World

credativ

- Distances of nodes - defined in firmware, numbers are theoretical

```
## continuation of command output - distances

[root@host5 node]# numactl --hardware
....
node distances:
node   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
```

PCIe devices and NUMA nodes

HOW 2026 Hello Open-source World

credativ

- Machine reported error in mapping PCIe devices to NUMA nodes - missing in firmware
- Firmware should define physical wiring of the PCIe devices to NUMA nodes
- Kylin cmdline compensated it with direct access to memory for specific PCI devices

```
[root@host5 node]# dmesg | grep -iE 'srat|slit|numa'
[    0.000643] mempolicy: Enabling automatic NUMA balancing. Configure with numa_balancing= or the kernel.↩
     numa_balancing sysctl
[    0.279337] pci_bus 0000:00: Unknown NUMA node; performance will be reduced

[root@host5 acpiinspect]# lspci -tv
-[0000:00]-+-00.0-[01-0c]--+-00.0-[02-0c]--+-01.0-[03-04]--+-00.0  Intel Corporation 82599ES 10-Gigabit SFI/↩
     SFP+ Network Connection
           |               |               |               \-00.1  Intel Corporation 82599ES 10-Gigabit SFI/↩
     SFP+ Network Connection
           |               |               +-08.0-[05]--
           |               |               +-09.0-[06-07]--+-00.0  Intel Corporation I350 Gigabit Network ↩
     Connection
           |               |               +-0a.0-[08]----00.0  Marvell Technology Group Ltd. 88SE9230 PCIe ↩
     2.0 x2 4-port SATA 6 Gb/s RAID Controller
           |               |               +-0b.0-[09]----00.0  Texas Instruments TUSB73x0 SuperSpeed USB 3.0↩
      xHCI Host Controller
....
[root@host5 ~]# cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-4.19.90-89.11.v2401.ky10.aarch64 root=/dev/mapper/klas-root ro rd.lvm.lv=klas/root rd.lvm↩
     .lv=klas/swap video=VGA-1:640x480-32@60me rhgb quiet console=tty0 crashkernel=1024M,high smmu.bypassdev↩
     =0x1000:0x17 smmu.bypassdev=0x1000:0x15 video=efifb:off audit=0
```

Memory Usage per NUMA Node

HOW 2026 Hello Open-source World

credativ

```
[root@host5 ~]# numastat -m

Per-node system memory usage (in MBs):
                          Node 0          Node 1          Node 2          Node 3
                 --------------- --------------- --------------- ---------------
MemTotal                14403.38            0.00            0.00        16362.81
MemFree                   989.69            0.00            0.00         1351.12
MemUsed                 13413.69            0.00            0.00        15011.69
Active                   6717.44            0.00            0.00        12015.12
Inactive                 5861.06            0.00            0.00         1811.69
Active(anon)             6553.62            0.00            0.00        11979.25
Inactive(anon)           5388.62            0.00            0.00         1761.31
Active(file)              163.81            0.00            0.00           35.88
Inactive(file)            472.44            0.00            0.00           50.38
Unevictable                 0.00            0.00            0.00            9.75
Mlocked                     0.00            0.00            0.00            9.75
Dirty                       0.25            0.00            0.00            0.06
Writeback                   0.00            0.00            0.00            0.00
FilePages                4780.56            0.00            0.00         3964.94
Mapped                   3460.81            0.00            0.00         3847.94
AnonPages                7798.56            0.00            0.00         9874.06
Shmem                    4113.00            0.00            0.00         3806.00
KernelStack                10.03            0.00            0.00            4.62
PageTables                 18.69            0.00            0.00           27.81
NFS_Unstable                0.00            0.00            0.00            0.00
Bounce                      0.00            0.00            0.00            0.00
WritebackTmp                0.00            0.00            0.00            0.00
....
```

Memory Usage per NUMA Node

HOW 2026 Hello Open-source World

credativ

```
[root@host5 ~]# numastat -n

Per-node numastat info (in MBs):
                          Node 0          Node 1          Node 2          Node 3
                 --------------- --------------- --------------- ---------------
Numa_Hit              19782858.62           18.56           18.56      4729650.81
Numa_Miss              1806725.19            0.00            0.00       508884.31
Numa_Foreign             49539.56            0.00            0.00      1162817.62
Interleave_Hit              95.69            0.00            0.00           87.19
Local_Node            19782241.81           18.00           18.00      3862957.81
Other_Node             1807342.00            0.56            0.56      1375577.31

                          Node 4          Node 5          Node 6          Node 7
                 --------------- --------------- --------------- ---------------
Numa_Hit                  18.56      2723213.56      3639347.12           18.56
Numa_Miss                  0.00       296391.00       937956.81            0.00
Numa_Foreign               0.00      1315067.81      1022532.31            0.00
Interleave_Hit             0.00           92.38           86.69            0.00
Local_Node                18.00       481743.56      2194208.75           18.00
Other_Node                 0.56      2537861.00      2383095.19            0.56

                           Total
                 ---------------
Numa_Hit              30875144.38
Numa_Miss              3549957.31
Numa_Foreign           3549957.31
Interleave_Hit             361.94
Local_Node            26321223.94
```

Distribution of buffers on NUMA nodes

HOW 2026 Hello Open-source World

credativ

```
SELECT numa_node,
       count(*) AS pages_allocated,
       pg_size_pretty(count(*) * (current_setting('block_size')::integer)) AS total_node_allocation,
       round(count(*) * 100.0 / sum(count(*)) over (), 2) AS pct_distribution
FROM pg_buffercache_numa
GROUP BY numa_node
ORDER BY numa_node;

 numa_node | pages_allocated | total_node_allocation | pct_distribution
-----------+-----------------+-----------------------+------------------
         0 |         1439368 | 11 GB                 |            27.45
         3 |          222256 | 1736 MB               |             4.24
         5 |         1938128 | 15 GB                 |            36.97
         6 |         1643128 | 13 GB                 |            31.34
```

Buffers Exceeding NUMA node

HOW 2026 Hello Open-source World

credativ

```
SELECT
    numa_node as "Physical NUMA Node",
    COUNT(*) as "8KB Buffer Pages",
    pg_size_pretty(COUNT(*) * 8192::bigint) as "Memory Volume"
FROM pg_buffercache_numa
GROUP BY numa_node
ORDER BY numa_node;

-- situation after restart - buffers exist mostly as virtual memory

 Physical NUMA Node | 8KB Buffer Pages | Memory Volume
--------------------+------------------+---------------
                  5 |          1991188 | 15 GB
                    |          3251692 | 25 GB

---> Attempt to physically fill all shared buffers crashed PostgreSQL
```

Relations on NUMA nodes

HOW 2026 Hello Open-source World

credativ

```
SELECT c.relname AS relation_name,
       n.numa_node,
       count(*) AS buffers,
       pg_size_pretty(count(*) * (current_setting('block_size')::integer)) AS memory_on_node
FROM pg_buffercache b
JOIN pg_buffercache_numa n ON b.bufferid = n.bufferid
JOIN pg_class c ON b.relfilenode = c.relfilenode
WHERE b.reldatabase IN (0, (SELECT oid FROM pg_database WHERE datname = current_database()))
  -- Optional: Filter for a specific problematic table
  -- AND c.relname = 'your_target_table_name'
GROUP BY c.relname, n.numa_node
ORDER BY c.relname, n.numa_node;
```

relation_name	numa_node	buffers	memory_on_node
large_random_data	0	1429167	11 GB
large_random_data	3	182707	1427 MB
large_random_data	5	1938122	15 GB
large_random_data	6	1643128	13 GB
numa_chain	0	10006	78 MB
numa_chain	3	39382	308 MB
numa_chain_pkey	3	4	32 kB
pg_aggregate	0	1	8192 bytes
pg_aggregate	3	1	8192 bytes

Quiz Time

The Alchemy of Shared Buffers - Workshop quiz for HOW2026 conference

forms.cloud.microsoft/e/UPUS00hcWd

HOW 2026
Hello Open-source World

Thank you for your attention!

credativ

Questions?

All my slides

Recorded talks