credativ

PostgreSQL Connections Memory Usage

How Much, Why and When?
On Debian/Ubuntu - x86-64 architecture

Josef Machytka <josef.machytka@credativ.de>

2025-05-08/09 - PostgreSQL Conference Germany 2025

credativ GmbH

- 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 and Proxmox
- Open-Source databases with PostgreSQL
- DevSecOps with Ansible, Puppet, Terraform and others
- Since 2025 independent owner-managed company again

Josef Machytka

credativ

- Professional Service Consultant - PostgreSQL specialist at credativ GmbH
- 30+ years of experience with different databases.
- PostgreSQL (12y), BigQuery (7y), Oracle (15y), MySQL (12y), Elasticsearch (5y), MS SQL (5y).
- 10+ years of experience with Data Ingestion pipelines, Data Analysis, Data Lake and Data Warehouse
- 2 years of practical experience with different LLMs / AI including their architecture and principles.
- From Czechia, living now 11 years in Berlin.

  - LinkedIn: linkedin.com/in/josef-machytka
  - ResearchGate: researchgate.net/profile/Josef-Machytka
  - Academia.edu: netapp.academia.edu/JosefMachytka
  - Medium: medium.com/@josef.machytka
  - sessionize: sessionize.com/josefmachytka

Table of contents

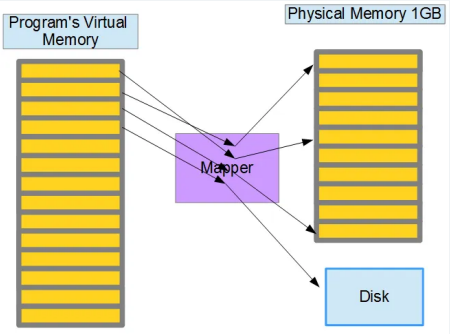
AI images created by the author
using DeepDreamGenerator
or ChatGPT DALL-E

credativ

Linux Memory Management

Memory Management on Linux x86-64 Architecture

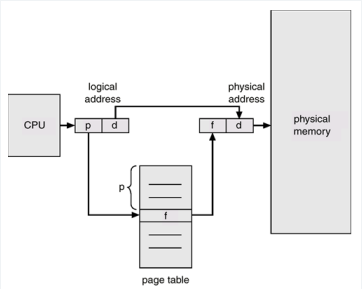
- Memory must be shared efficiently between processes
- Empty memory is wasted memory
- Linux uses virtual memory to abstract physical memory
- Allows to use more memory than physically available
- Gives each process an illusion of having all memory to itself
- Each process has its own virtual address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



(Image from the article
Virtual Memory & Physical Memory)

Memory Management Unit (MMU)

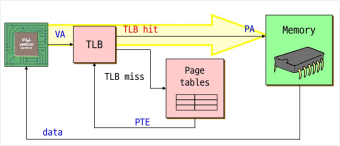
- Hardware unit, translates virtual to physical addr
- 64-bit x86 architecture - page 4 KB, 2 MB or 1 GB
- For quick access - Translation Lookaside Buffer (TLB)
- If not in TLB, CPU must ”walk” through page tables (slow)



(Image from the article
MMU & Virtual Memory)

Huge Pages

- Managing many 4 KB small pages causes significant overhead
- Huge pages 2 MB or 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
- But huge pages cannot be swapped out



(Image from the article Virtual Memory)

Transparent Huge Pages

credativ

- Transparent Huge Pages (THP) is name of Linux kernel feature
- Kernel can automatically reallocate many small pages into huge page and back
- But PostgreSQL docu still warns, THP has been known to cause performance degradation

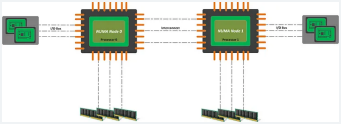
- But we can use explicit HP - settings *huge_pages* and *huge_pages_size*
- huge_pages - (try/on/off) - ”on” can block postmaster start if it does not work
- PostgreSQL 17 reports if huge pages are used - new read-only param *huge_pages_status*

```
cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never

echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

None Uniform Memory Access (NUMA)

- NUMA is a computer memory design used in multiprocessing
- It separates memory into different banks
- Each bank is connected to a different processor
- It reduces contention for memory access
- Increases memory bandwidth, reduces latency
- It is used in modern servers with multiple CPUs
- PostgreSQL 18 got initial support for NUMA awareness
- Should improve performance on multi-node/multi-socket servers



(Image from the article Mastering NUMA Nodes in Linux)

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

Anonymous Memory vs File-backed Memory

- *Anonymous memory* - not associated with a file, can be swapped out, cannot be reclaimed by OS
- *File-backed memory* - associated with a file, can be reclaimed by OS

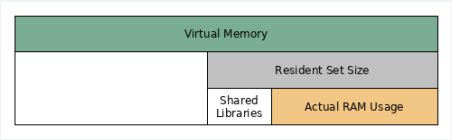
- Man page of ”top” command shows this schema:

```
                          Private | Shared
                     1            |          2
  Anonymous   . stack             |
              . malloc()          |
              . brk()/sbrk()      | . POSIX shm*
              . mmap(PRIVATE, ANON) | . mmap(SHARED, ANON)
             ---------------------+---------------------
              . mmap(PRIVATE, fd) | . mmap(SHARED, fd)
File-backed   . pgms/shared libs  |
                     3            |          4
```

Memory Sizes based on states of allocation

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



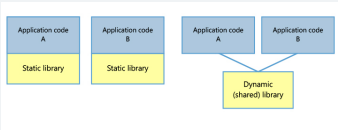
(Image from the article Virtual Memory vs. Resident Set Size)

Shared Memory vs Shared Libraries

credativ

- *Shared Memory* - memory that can be accessed by multiple processes
- For sharing large amounts of data between processes
- Shared memory segment is created by one process and attached by others

- *Shared Libraries* - code that can be used by multiple processes
- Loaded into memory based on references in the code
- Once loaded, can be reused by other processes requiring the same library
- But shared library can be also used by only one process
- Shared library can use shared memory to share data between processes



(Image from the article How to integrate third-party library)

Memory Overcommitment

- Linux allows to allocate more memory than physically available
- Assumption - not all processes will use all memory they requested
- Parameter cat */proc/sys/vm/overcommit_memory* controls behavior:
  - 0 - heuristic overcommitment (default)
  - 1 - always overcommit
  - 2 - never overcommit

Out Of Memory (OOM) Killer

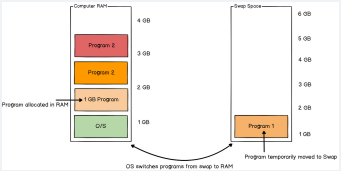
- OOM killer is active when overcommit is enabled
- Scores processes - how much system would gain by killing it
- Score in */proc/$PID/oom_score* file
- Can be influenced by */proc/$PID/oom_score_adj* file
- (from -1000 to +1000, lower score = lower chance of killing)

- /lib/systemd/system/postgresql@.service - OOMScoreAdjust=-900
- Comment: To prevent OOM killer from choosing the postmaster
- Individual backends will reset the score to 0

Is SWAP Good, Bad, or just Ugly?

- Swap is a part of disk used as an extension of physical memory
- If physical memory is under pressure, system can swap out 4 KB pages
- Prevents OOM killer from killing processes

- On old hard drives, swapping was slow, caused performance issues
- But modern, locally attached SSDs are very fast
- Swapping is not so bad anymore, maybe just a bit ugly?
- There are strong opinions on both sides...



(Image from the article Swap space)

credativ

How To Measure Memory Usage

Command ”free” – Overview of Memory Usage

- *total*, *used*, *free* are quite self-explanatory
- *free* - MemFree and SwapFree in /proc/meminfo
- *shared* - used by tmpfs (temporary file system - the biggest part) and shared memory segment
- *buff/cache* - used by kernel buffers and page cache, can be reclaimed
- *available* - available for new processes: free + buff/cache - minimum for system
- Parameter -s [seconds] refreshes the output every [seconds]
- -v displays memory commit limit, amount of committed/uncommitted memory

```
[redacted]:/# cat /proc/sys/vm/overcommit_memory
0
[redacted]:/# free -hv
               total        used        free      shared  buff/cache   available
Mem:            31Gi        12Gi       9.2Gi       2.2Gi        12Gi        18Gi
Swap:           31Gi          0B        31Gi
Comm:           46Gi        76Gi       -29Gi
```

”top” Command – Memory Usage by Process

credativ

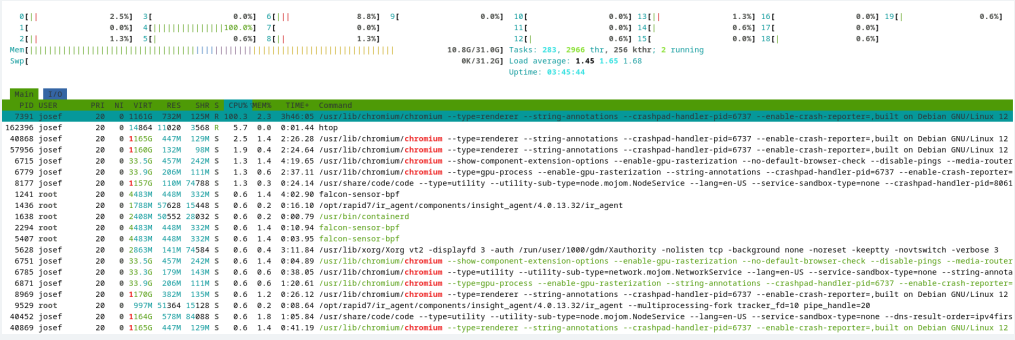
- *VIRT* - Virtual memory size - all memory requested by the process (data, code, shared)
- *RES* - Resident memory size - physical memory used by the process + shared libraries
- *SHR* - Shared memory size - shared with other processes (shared libraries, shared memory)
- Parameter -c shows command line arguments
- *-E [k/m/g]* - sets memory units for summary area
- -e *[k/m/g]* - sets memory units for task area
- -p *[PID]* - shows data only for the specified process

```
top - 12:26:00 up  3:33,  2 users,  load average: 1.74, 1.63, 1.73
Tasks: 547 total,   2 running, 545 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.7 us,  0.2 sy,  0.0 ni, 94.0 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 32521804 total,  7887256 free, 14372068 used, 13246060 buff/cache
KiB Swap: 32763900 total, 32763900 free,        0 used. 18149736 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   7391 josef     20   0 1160.4g 746472 129088 R  99.7   2.3 214:02.96 /usr/lib/chromium/chromium --type=renderer --string
   5888 josef     20   0 7113932 429340 123684 S   9.3   1.3   3:53.85 /usr/bin/gnome-shell
   1241 root      20   0 4591504 458236 340592 S   1.0   1.4   3:55.57 falcon-sensor-bpf
   6715 josef     20   0   32.8g 468948 247848 S   1.0   1.4   4:09.66 /usr/lib/chromium/chromium --show-component-extens
  40868 josef     20   0 1163.6g 444232 132500 S   1.0   1.4   2:18.31 /usr/lib/chromium/chromium --type=renderer --string
  57956 josef     20   0 1160.3g 136084 100640 S   1.0   0.4   2:12.92 /usr/lib/chromium/chromium --type=renderer --string
```

”htop” Command - Interactive Process Viewer

- Nice interactive process viewer
- But still shows only virtual memory size and resident memory size



”pg_top” Command – like ”top” for PostgreSQL

credativ

- Shows only postgres processes
- But still shows only virtual memory size and resident memory size

```
last pid:  5883;  load avg:  0.07,  0.05,  0.08;       up 0+00:47:47
7 processes: 5 other background task(s), 1 idle, 1 active
CPU states:  1.1% user,  0.0% nice,  0.4% system, 98.5% idle,  0.0% iowait
Memory: 11G used, 261M free, 0K shared, 51M buffers, 10G cached
DB activity:   0 tps,  0 rollbs/s,   0 buffer r/s, 100 hit%,    101 row r/s,    0 row w/s s
DB I/O:     0 reads/s,     0 KB/s,     0 writes/s,     0 KB/s
Swap: 256K used, 4096M free, 60K cached, 0K in, 0K out

    PID USERNAME    SIZE   RES STATE   XTIME  QTIME  %CPU LOCKS COMMAND
   5884 postgres   3277M   22M active   0:00   0:00   0.0     8 postgres: 17/main: postgres postgres [local] idle
   4321            3276M 8828K          0:00   0:00   0.0     0 postgres: 17/main: autovacuum launcher
   4320            3274M   21M          0:00   0:00   0.0     0 postgres: 17/main: walwriter
   4317            3274M   32M          0:00   0:00   0.0     0 postgres: 17/main: checkpointer
   4318            3274M   30M          0:00   0:00   0.0     0 postgres: 17/main: background writer
   4322 postgres   3276M 7932K          0:00   0:00   0.0     0 postgres: 17/main: logical replication launcher
   4595 postgres   3417M 1850M idle     0:00   0:00   0.0     0 postgres: 17/main: postgres postgres [local] idle
```

”smem” Command - USS/PSS Memory Usage

credativ

- *USS* - Unique Set Size - physical memory unique to the process
- *PSS* - Proportional Set Size - USS + shared memory portion
- *RSS* - Resident Set Size - physical memory used by the process + shared libraries

- *-s [uss/pss/rss]* - sorts by USS, PSS or RSS memory
- *-r* - sort in reverse order
- *-u* - shows memory summary usage per user
- *-w* - show system wide memory usage summary

```
  PID User     Command                         Swap      USS      PSS      RSS
36977 josef    /home/josef/.vscode/extensi      700        4        4        8
37138 josef    /usr/bin/bash --init-file /     1904        4       16     2276
38923 josef    /usr/bin/bash --init-file /     1920        4       16     2240
228719 josef    /usr/bin/bash --init-file /     1920        4       16     2256
```

System sources for memory usage

- proc filesystem - /proc/meminfo, /proc/[PID]/status, /proc/[PID]/smaps
- For our tests smaps fits best, contains multiple entries for each memory region
- Header contains memory region start and end address, permissions, path to the file
- Some memory regions do not contain any path

```
557b3de3c000-557b3df0e000 r--p 00000000 fd:02 14426095                   /usr/lib/postgresql/16/bin/postgres
Size:                840 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
Rss:                 544 kB
Pss:                 272 kB
Pss_Dirty:             0 kB
Shared_Clean:        544 kB
Shared_Dirty:          0 kB
Private_Clean:         0 kB
Private_Dirty:         0 kB
Referenced:          544 kB
Anonymous:             0 kB
LazyFree:              0 kB
AnonHugePages:         0 kB
ShmemPmdMapped:        0 kB
FilePmdMapped:         0 kB
Shared_Hugetlb:        0 kB
Private_Hugetlb:       0 kB
Swap:                  0 kB
SwapPss:               0 kB
Locked:                0 kB
```

credativ

PostgreSQL Memory Objects

Shared Buffers

- All processes need some shared memory
- Shared buffers - the biggest & most discussed
- Cache of tables and indexes data blocks
- In-memory copy of blocks, shared among connections
- Kept based on frequency of access
- Recommended 25% of the available memory
- `pg_buffercache` monitors shared buffers
- Allocated first only as a virtual memory

Other Shared Memory Objects

- Allocated on startup for tracking locks, WAL buffer, SLRU (Simple Least Recently Used) buffers
- 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)
- Includes shared buffers, lock table, WAL buffers, SLRU buffers etc.
- Can be exhausted during some operations -> ”out of shared memory” error
- */dev/shm* memory for communication between parallel workers
- If exhausted -> ”could not resize shared memory segment” error (in docker default 64MB)
- Change in settings for shared memory requires restart

```
postgres=# select name, setting from pg_settings where name like '%shared_memory%' order by name;
                name                | setting
------------------------------------+---------
 dynamic_shared_memory_type         | posix
 min_dynamic_shared_memory          | 0
 shared_memory_size                 | 8423
 shared_memory_size_in_huge_pages   | 4212
 shared_memory_type                 | mmap
(5 rows)
```

credativ

PostgreSQL Connection Memory Usage

PostgreSQL Connections

- Each connection is an independent process
- With its own isolated memory
- Great for stability & security, but resource-intensive
- Processes chosen for higher stability in the 1990s
- Reliable threads implemented in Linux 2.6 (2003/2004)
- Threads are now reliable and considered lightweight
- Ongoing discussion about switch to threads

How Much Memory PostgreSQL Connection Uses?

- PC 32GB memory, PostgreSQL 17, shared_buffers=8GB, effective_cache_size=24GB, work_mem=64MB
- Connected with psql, connection is newly created, no command issued yet
- Python script with *psutil* library calls *memory_full_info()*
- When I closed the connection and opened again, I got similar numbers

```
## 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 - psutil.memory_full_info()
PID: 190747, Command: postgres: postgres postgres 172.18.0.1(40278) idle
    rss:          19.8 MB
    vms:        8497.6 MB
    shared:       16.5 MB
    text:          5.4 MB
    lib:           0.0 MB
    data:          3.6 MB
    dirty:         0.0 MB
    uss:           2.2 MB
    pss:           8.4 MB
    swap:          0.0 MB
```

How smaps Looks Like for PostgreSQL Connection?

- Another Python script used to parse and pivot the /proc/PID/smaps file, to show the memory usage
- Here is detailed view - showed 42 different /usr/lib/x86_64-linux-gnu/ libraries
- And many small regions without paths -> summarized together as [anonymous]

```
## 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/16/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/16/lib/auto_explain.so	20	8	0	0	0	8	0	0	0	0	5
/usr/lib/postgresql/16/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

credativ

- Let's run some heavy aggregations over the table not fitting into memory
- Memory is 32 GB, table has 38 GB, shared_buffers= 8 GB, work_mem= 64 MB
- No parallelism - max_parallel_workers_per_gather = 0
- What we see after the query execution (pg_buffercache_summary shows shared buffers fully used)

```
## 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

## python script output - psutil.memory_full_info()
PID: 190747, Command: postgres: postgres postgres [local] idle
    rss:    8344.3 MB
    vms:    8535.2 MB
 shared:    8340.5 MB
   text:       5.6 MB
    lib:       0.0 MB
   data:       3.9 MB
  dirty:       0.0 MB
    uss:    8196.1 MB
    pss:    8251.4 MB
   swap:       1.0 MB
```

Let’s Run Some Heavy Query

- We must 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/16/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/16/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
```

What about Parallelism?

- In the first session I switched off parallelism
- Now I start new session use max_parallel_workers_per_gather=4
- Process started 4 parallel workers, but each attached only part of shared buffers
- Therefore where query ended, the main process had only part of shared buffers attached

```
## top command - after run ended
    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

## smaps - after run ended
Path                                          Size        Rss        Pss  Pss_Dirty  Shr_Clean  Shr_Dirty  Prv_Clean  Prv_Dirty      Swap  SwapPss  Cnt
-------------------------------------------------------------------------------------------------------------------------------------------------------
/usr/lib/postgresql/16/bin/postgres           9296       4864       1711         65       4220        104        488         52        64        8    5
[anonymous]                                   1192        552        484        484          0         76          0        476        40        3   21
[heap]                                        2012       1556       1423       1423          0        152          0       1404       204       27    2
/dev/shm/                                     2356        192        115        115          0        148          0         44       988        0    4
/usr/lib/postgresql/16/lib/                     84         56         46         16          4          8         28         16         0        0   15
/dev/zero (deleted)                         8624208    4927884    2431359    2431359          0    4927592          0        292     40764        0    1
/usr/lib/locale/locale-archive                2980         64         64          0          0          0         64          0         0        0    1
/usr/lib/x86_64-linux-gnu/                   60520       1144        397         35        780        244         96         24      1040      109  205
/SYSV00ce5741 (deleted)                          4          0          0          0          0          0          0          0         4        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                                      8702808    4936360    2435643    2433541       5008    4928324        676       2352     43104      147  258
```

How Much Memory is Really Used?

- I did some playing with multiple sessions with/without parallelism
- Let's now check "top" and "free" commands

```
## 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

## free command
               total        used        free      shared  buff/cache   available
Mem:            31Gi        15Gi       698Mi       9.3Gi        24Gi        15Gi
Swap:           31Gi       4.5Gi        26Gi
Comm:           46Gi        67Gi       -21Gi
```

But How Much Memory are Connections Really Using?

credativ

- Let's dive into smaps of all these sessions and do some math - sizes are in KBs

```
## 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
```

credativ

PostgreSQL work_mem mystery

Where is work_mem hiding?

credativ

- Usually seen as the most important memory setting
- Some claim it is the memory used by a single connection
- But our previous tests did not show it
- Not fixed number, it is limit
- Docs: *"maximum amount of memory to be used by a query"*
- Setting *hash_mem_multiplier* (default 2) for hash joins
- So, where it is hidden in the smaps paths and numbers?
- We must scrape smaps in loop during the query execution
- Different data and queries tested, best results with JSONB

Tests with JSONB Data of GitHub Events

credativ

- I tested different data and queries, but best results with JSONB data from GitHub events
- Table size 38 GB, 17.5 million rows, 1/3 of JSONB records toasted, lz4 compression, PG 17.2

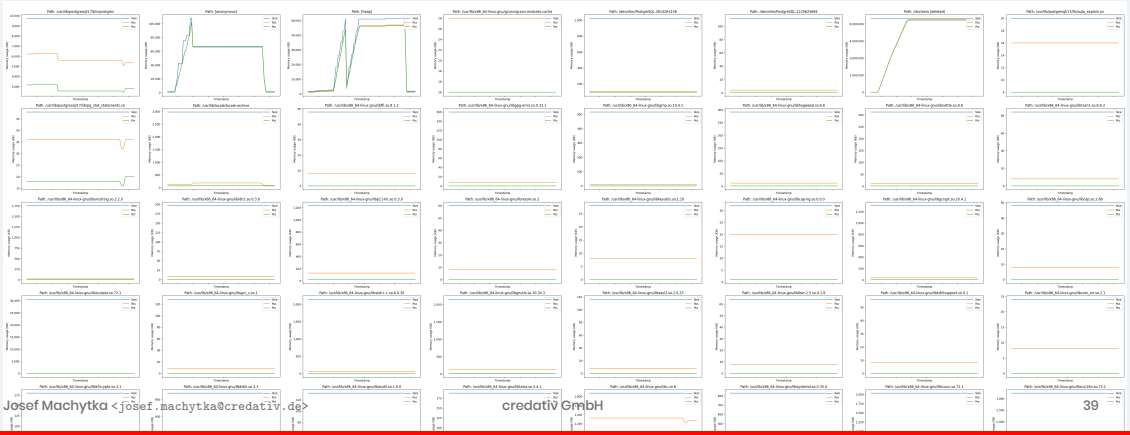
```
-- testing query
SELECT
    SUBSTR(jsonb_data->>'created_at'::TEXT,1,10) as created_at,
    count(*) as cnt
FROM github_events
GROUP BY 1 ORDER BY 1;

-- testing data
{   "id": "26167585827",
    "repo": {   "id": 581592468, "url": "https://api.github.com/repos/tiwabs/tiwabs_audio_door_tool",
                "name": "tiwabs/tiwabs_audio_door_tool" },
    "type": "PushEvent",
    "actor": {  "id": 48737497, "url": "https://api.github.com/users/tiwabs",
                "login": "tiwabs", "avatar_url": "https://avatars.githubusercontent.com/u/48737497?",
                "gravatar_id": "", "display_login": "tiwabs"  },
    "public": true,
    "payload": {"ref": "refs/heads/master", "head": "3ca247941f269bcedeb17e5b12e9b3b74b1c4da2",
                "size": 1, "before": "0dd5471667b12084b8fc88b1bca299780382d50a",
                "commits": [ {   "sha": "3ca247941f269bcedeb17e5b12e9b3b74b1c4da2",
                            "url": "https://api.github.com/repos/tiwabs/....12e9b3b74b1c4da2",
                            "author": { "name": "Tiwabs", "email": "mrskielz@gmail.com" },
                            "message": "fix(export): export nametable if export succed",
                            "distinct": true } ],
                "push_id": 12149772587, distinct_size": 1 },
    "created_at": "2023-01-01T13:39:55Z" }
```

Logging smaps for PostgreSQL process

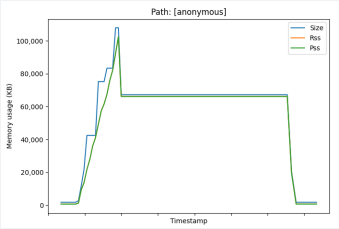
credativ

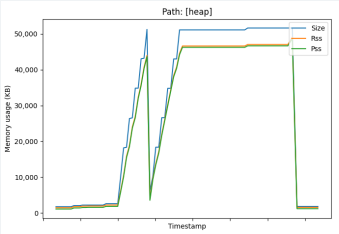
- Python script logs smaps data for PostgreSQL connections, roughly every 0.5 seconds
- RAM 32 GB, table 38 GB, shared_buffers 8 GB, work_mem 64 MB, toasted JSON, no parallel workers
- Query runs approx 70 seconds, after that we plot grid (Virt Size, RSS, PSS) for all paths

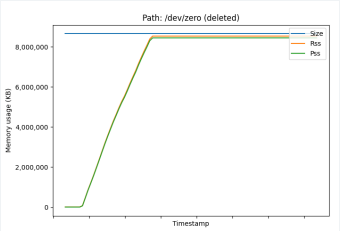


Analyzing logged smaps data

- All paths are mostly flat and un-interesting, except for 3 - [heap], [anonymous] and /dev/zero
- RAM 32 GB, table 38 GB, shared_buffers 8 GB, work_mem 64 MB, toasted JSON, no parallel workers

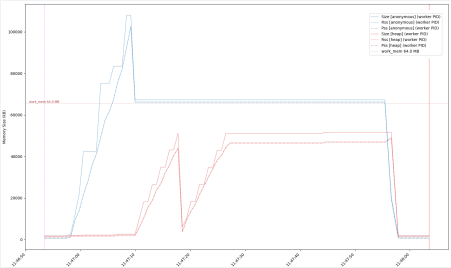


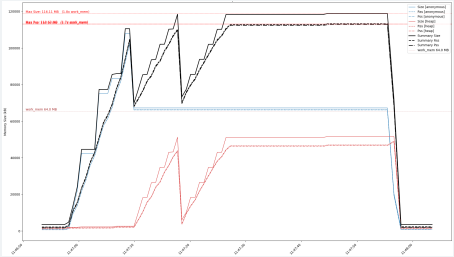




Smaps paths [heap] & [anonymous]: work_mem 64 MB

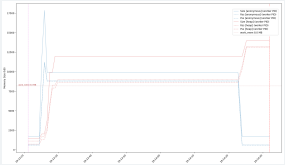
- During query executions, we see significant changes in *[heap]* and/or *[anonymous]* regions
- RAM 32 GB, table 38 GB, shared_buffers 8 GB, work_mem 64 MB, toasted JSON, no parallel workers
- memory usage exceeding work_mem 64 MB approx 1.7 times (RSS/PSS)





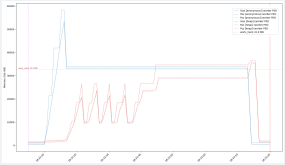
Smaps data - single process, work_mem 8 / 32 / 64 / 128 MB

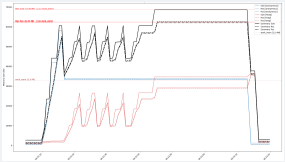
work_mem = 8 MB



Max stacked RSS = 17.5 MB
(2.2x work_mem)
Sort Method: external merge
Disk: 307960kB

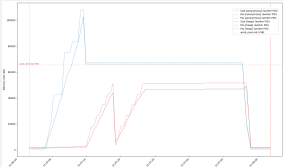
work_mem = 32 MB

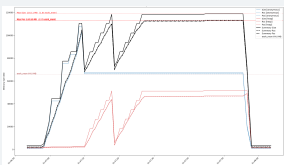




Max stacked RSS = 61 MB
(1.9x work_mem)
Sort Method: external merge
Disk: 307864kB

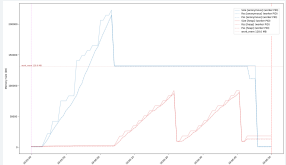
work_mem = 64 MB

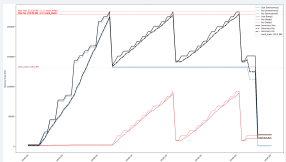




Max stacked RSS = 110.6 MB
(1.7x work_mem)
Sort Method: external merge
Disk: 307822kB

work_mem = 128 MB

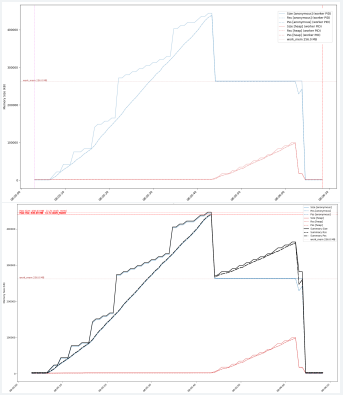




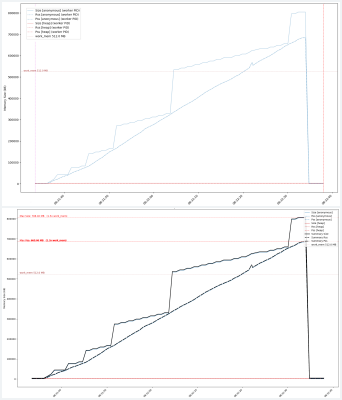
Max stacked RSS = 216 MB
(1.7x work_mem)
Sort Method: external merge
Disk: 307792kB

Smaps paths [heap] & [anonymous]: work_mem 256 / 512 MB

work_mem = 256 MB
Sort Method: external merge Disk: 307776kB
Max stacked RSS = 430 MB
(1.7x work_mem)

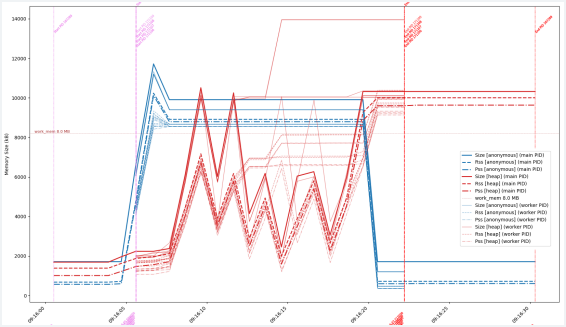


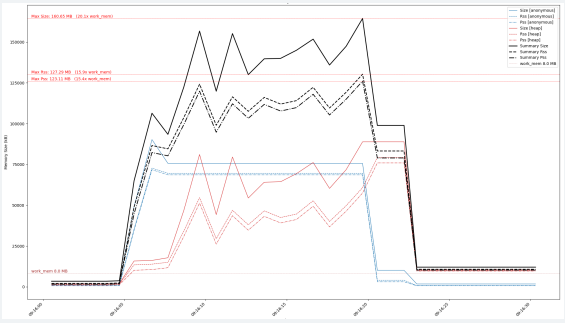
work_mem = 512 MB
Sort Method: quicksort Memory: 524288kB
Max stacked RSS = 670 MB
(1.3x work_mem)



What about parallelism - work_mem 8 MB, 8 parallel workers

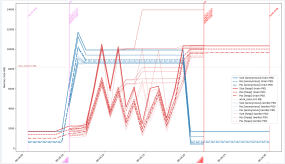
- work_mem = 8 MB, max_parallel_workers_per_gather = 8 - Workers Launched: 7, runtime 18 sec (vs 70 sec)
- each process (main + 7 workers) doing external merge on disk: 8 x 38 MB
- max stacked RSS = 128 MB (16x work_mem)





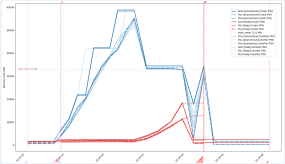
Smaps data - main + 7 workes, work_mem 8 / 32 / 64 / 128 MB

work_mem = 8 MB



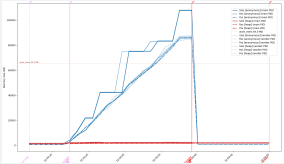
Max stacked RSS = 128 MB
(16x mork_mem)
Disk sorts 8x 38 MB

work_mem = 32 MB



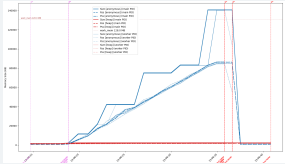
Max stacked RSS = 436 MB
(13.6x work_mem)
Disk sort 8x 38 MB

work_mem = 64 MB



Max stacked RSS = 684 MB
(10.7x work_mem)
Memory sort 8x 64 MB

work_mem = 128 MB



Max stacked RSS = 685 MB
(5.4x work_mem)
Memory sort 8x 96 MB

credativ

Bonus: Query with 2 ORDER BY clauses

Testing query with 2 ORDER BY clauses

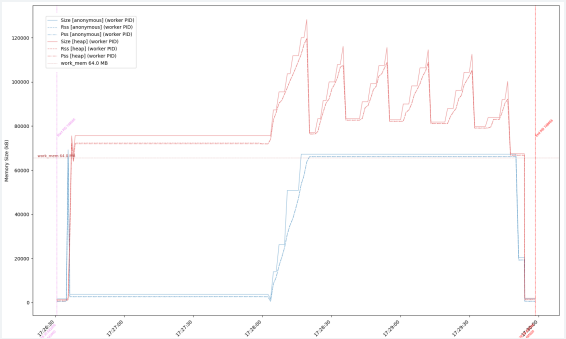
credativ

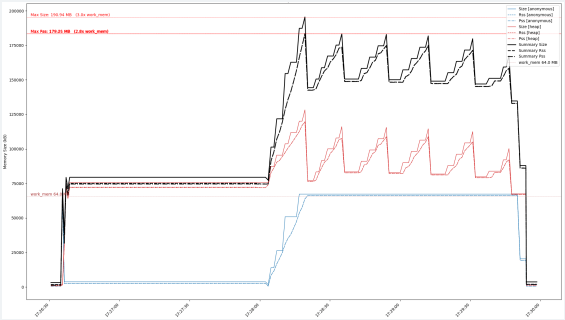
- Query with 2 ORDER BY clauses - 1st ORDER BY in CTE, 2nd ORDER BY in main query
- Table size 38 GB, 17.5 million rows, 1/3 of JSONB records toasted, lz4 compression, PG 17.2

```
WITH ordered as (
    SELECT jsonb_data
    FROM github_events
    ORDER BY jsonb_data->>'created_at'::text)   /* <<-- 1st ORDER BY */
SELECT
    substr(jsonb_data->>'created_at'::text,1,10) as created_at,
    count(*) as cnt
FROM ordered
GROUP BY 1
ORDER BY 1;  /* <<-- 2nd ORDER BY */
```

Smaps paths [heap] & [anonymous]: work_mem 64 MB

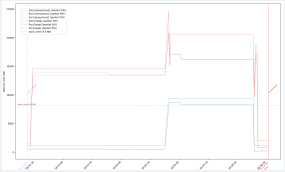
- Explain plan shows 2 big disk sorts:
- CTE part: Sort Method: external merge Disk: 13,615,648 kB
- Main query part: Sort Method: external merge Disk: 307,816 kB
- Max stacked RSS = 179 MB (2.8x work_mem)

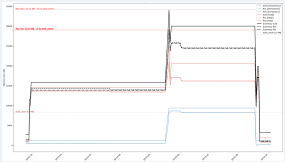




Smaps data - single process, work_mem 8 / 32 / 64 / 128 MB

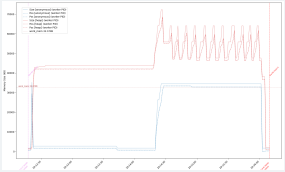
work_mem = 8 MB

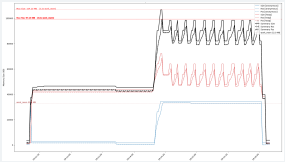




Max stacked RSS = 28.4 MB
(3.6x work_mem)

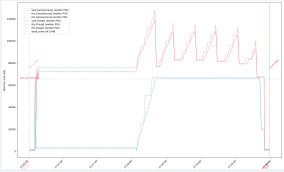
work_mem = 32 MB

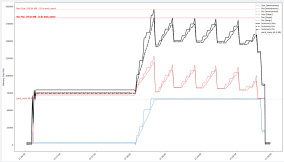




Max stacked RSS = 97 MB
(3x work_mem)

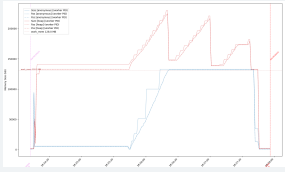
work_mem = 64 MB

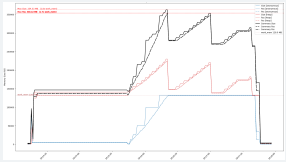




Max stacked RSS = 179 MB
(2.8x work_mem)

work_mem = 128 MB

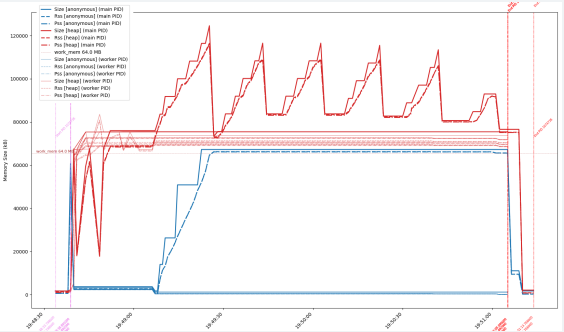


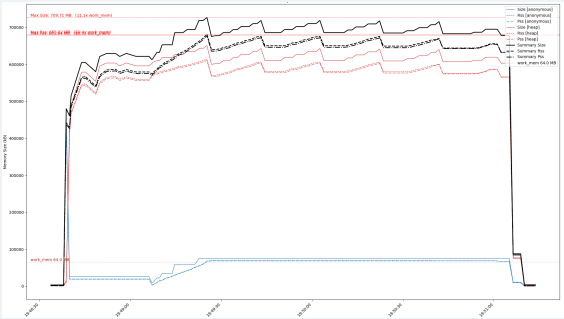


Max stacked RSS = 346 MB
(2.7x work_mem)

What about parallelism - work_mem 64 MB, main + 7 workers

- work_mem = 64 MB, max_parallel_workers_per_gather = 8 - Workers Launched: 7, runtime 2 minutes
- each process (main + 7 workers) doing external merge on disk: 8 x 1.6 GB -> approx 13 GB
- Parallel workers operate mainly with [heap], [anonymous] used almost exclusively only in main
- Max stacked RSS = 666 MB (10.4x work_mem)





credativ

Summary

What We Learned

- PostgreSQL connections are not so memory hungry as they seem
- Numbers in ”top” command contain attached shared buffers
- Work_mem is not a fixed per-connection memory allocation
- ”soft limit” - may not be fully used, but can be exceeded
- Also parallel workers can each exceed work_mem too
- Be very very cautious with calculating max_connections
- Just work_mem vs available memory is not enough

credativ

Questions?