What is an SLRU anyway?

Álvaro Herrera – Postgres developer, EDB

PGDay Austria 2025
Vienna
4 September 2025

Who am I?

- Álvaro Herrera, alvherre@kurilemu.de
- Postgres contributor since 2002
- Working as a Postgres developer for 2ndQuadrant/EDB since 2012
- Worked on a number of SLRU-related projects:
  - savepoints
  - multixacts
  - commit timestamps
- I also came up with autovacuum, background workers, BRIN indexes, and more
  - Feel free to grab hold of me on hallways to talk about stuff

Talk structure

1. Historical review of SLRU development
2. Report of an SLRU performance problem
3. Description of the solution

What **is** an SLRU?

- A very bad name for user-visible feature
- **S**imple **L**east **R**ecently **U**sed
- A mechanism to store transactional metadata
  - And things with similar behavior
- Examples:
  - Transaction commit/abort status
  - LISTEN / NOTIFY data
  - transaction commit times
- Keeps memory buffer of on-disk data

Historical review of SLRU development

Development History: 1. `pg_clog`

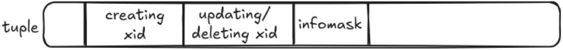
- Problem: after 4 billion transactions, your database ist total kaputt [1]
- Fix: Simplistic `pg_log` pseudo-relation is replaced with `pg_clog`
- Commit 2589735da08c:
  Replace implementation of pg_log as a relation accessed through the buffer manager with 'pg_clog', a specialized access method modeled on pg_xlog.
  Tom Lane

  Sat Aug 25 2001, Postgres 7.2

[1] Transaction ID wraparound: problem and proposed solution (Fri 03 Nov 2000)

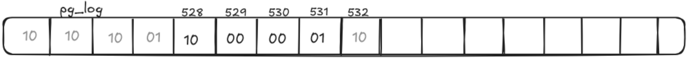
Transaction status storage

- Reading a tuple requires looking up status of its creating and deleting transactions



- Two bits per transaction id (xid):
  - 00 → “in-progress”
  - 01 → “aborted”
  - 10 → “committed”

How pg_clog works



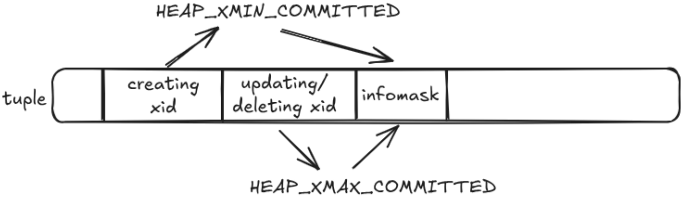
	pg_log			528	529	530	531	532								
10	10	10	01	10	00	00	01	10								

- 0x8000 (decimal 32768) transactions per 8kB page
- 8 in-memory pages store the status of $32768 * 8 = 262144 = 256k$ transactions
- 32 pages per file
  - after 1M transactions, a file can be removed
  - “wraparound” is now possible

How `pg_clog` works (2)

- At this point, LRU is an internal `pg_clog` implementation detail
- Harcoded buffer size of 8 pages
  - Much later, `pg_clog` was renamed `pg_xact`, commit 88e66d193fba (2017).

How pg_clog works (3)



- If transaction committed long enough ago, "hint bits" are set in the "infomask"
- When tuples are "hinted", no pg_clog lookups
- Old pg_clog segments no longer needed

Development History: 2. `slru.c`

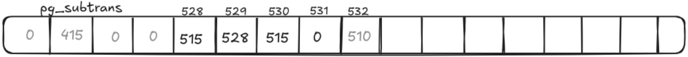
- My first big project: `SAVEPOINTS` (geb. “nested transactions”)
- slru.c was born from `pg_clog` to support this
- Commit 0abe7431c6d7:
  This patch extracts page buffer pooling and the simple least-recently-used strategy from clog.c into slru.c.
  Manfred Koizar (via Bruce Momjian)

  Wed Jun 11 2003, Postgres 7.4
- The term “slru” was invented at this point
- Nobody thought that this name would ever be exposed to users

Development History: 3. `pg_subtrans`

- Nested transactions
- Commit 573a71a5da70:
  Nested transactions.
  Álvaro Herrera, with some help from Tom Lane

  Thu Jul 1 2004, Postgres 8.0
- `pg_subtrans`: needed to store parent TransactionId for each *subtransaction*
- First user of `slru.c` outside of `pg_clog.c`

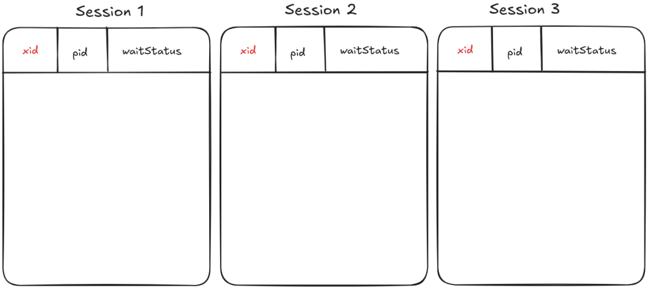


How does pg_subtrans work?

- 4 bytes per transaction (16x larger than pg_clog!)
- 8 pages of 8kB each have room for 16k transactions
- But what is pg_subtrans for?

What is pg_subtrans for?

Without subtransactions, answering whether a transaction is running is easy reading only memory:



Session 1

xid	pid	waitStatus

Session 2

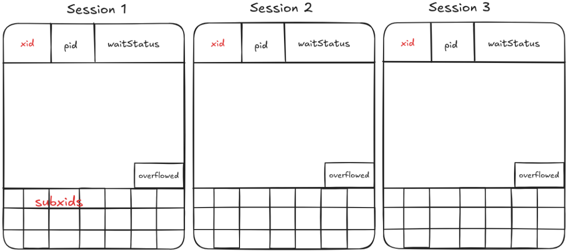
xid	pid	waitStatus

Session 3

xid	pid	waitStatus

What is pg_subtrans for? (2)

- pg_subtrans responds to “is transaction X running?” in presence of subtransactions
- ... only needed for transactions with >64 subtransactions

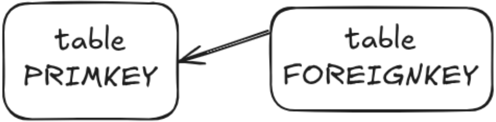


Development History: `pg_multixact`

- Commit d1e027221d02:
  Implement sharable row-level locks, and use them for foreign key references to eliminate unnecessary deadlocks.
  Álvaro Herrera and Tom Lane

  Thu Apr 28 2005, Postgres 8.1
- pg_multixact: A two-level mechanism to store variable-sized arrays using fixed-size `slru` addressing

foreign keys: avoiding SELECT FOR UPDATE

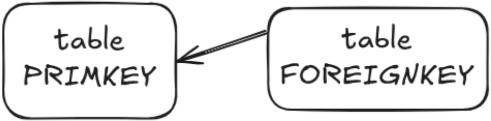


INSERT INTO foreignkey (...)

↳SELECT .. FROM primkey FOR UPDATE

tuple		creating xid	updating/ locking xid	infomask	

foreign keys: SELECT FOR SHARE to the rescue

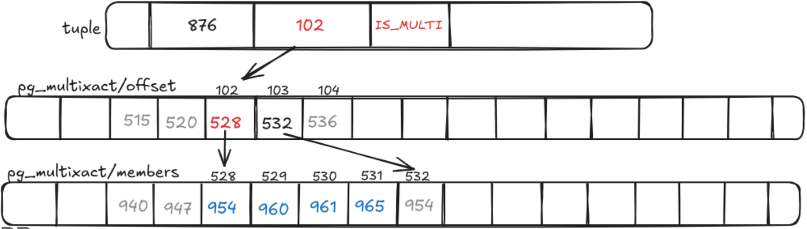


INSERT INTO foreignkey (...)

↳SELECT .. FROM primkey FOR ~~UPDATE~~ SHARE

How does pg_multixact work?

- Each MultiXactId is a pointer to pg_multixact/offset
- Each multixact offset is a pointer to pg_multixact/members
- We know how many members to read by reading the offset after ours



Development History 5: Variable sized SLRUs

- Commit 887a7c61f630: ↗
  Get rid of slru.c's hardwired insistence on a fixed number of slots per SLRU area. The number of slots is still a compile-time constant (someday we might want to change that), but at least it's a different constant for each SLRU area. Increase number of subtrans buffers to 32 based on experimentation with a heavily subtrans-bashing test case, and increase number of multixact member buffers to 16, since it's obviously silly for it not to be at least twice the number of multixact offset buffers.
  Tom Lane

  Tue Dec 6 2005, Postgres 8.2

Development History 5: Variable sized SLRUs

- Commit 887a7c61f630: ↗
  Get rid of slru.c's hardwired insistence on a fixed number of slots per SLRU area. The number of slots is still a compile-time constant (someday we might want to change that), but at least it's a different constant for each SLRU area. Increase number of subtrans buffers to 32 based on experimentation with a heavily subtrans-bashing test case, and increase number of multixact member buffers to 16, since it's obviously silly for it not to be at least twice the number of multixact offset buffers.
  Tom Lane

  Tue Dec 6 2005, Postgres 8.2

Development History 5: Variable sized SLRUs

- Commit 887a7c61f630:
  Get rid of slru.c's hardwired insistence on a fixed number of slots per SLRU area. The number of slots is still a compile-time constant (someday we might want to change that), but at least it's a different constant for each SLRU area. Increase number of subtrans buffers to 32 based on experimentation with a heavily subtrans-bashing test case, and increase number of multixact member buffers to 16, since it's obviously silly for it not to be at least twice the number of multixact offset buffers.
  Tom Lane

  Tue Dec 6 2005, Postgres 8.2

(Short) theory of operation

Why a small number of buffers? Here's how it works:

1. Scan linearly the array of buffers to see if one contains the page we want
2. If we find it, we're done
3. If not, the scan has chosen a "victim" buffer to evict (least recently used)
    1. Evict it, leaving buffer free
    2. Load our page onto our buffer
    3. Increment "recently used" counter
4. Now the page in buffer can be processed

(Short) theory of operation

Why a small number of buffers? Here's how it works:

1. Scan linearly the array of buffers to see if one contains the page we want
2. If we find it, we're done
3. If not, the scan has chosen a "victim" buffer to evict (least recently used)
    1. Evict it, leaving buffer free
    2. Load our page onto our buffer
    3. Increment "recently used" counter
4. Now the page in buffer can be processed

Development History 6: pg_notify

- Commit d1e027221d02:
  Replace the pg_listener-based LISTEN/NOTIFY mechanism with an in-memory queue.
  Joachim Wieland (via Tom Lane)
  Tue Feb 16 2010, Postgres 9.0
- This allowed NOTIFY to carry user-specified payload
- SLRU buffer of 8 pages
  - ... but pages only have to be retained until all backends read notification messages
  - ... which happens as soon as they run any command at all
  - Small chances of overflowing the buffer

Development History 7: pg_serial

- Commit dafaa3efb75c:
  Implement genuine serializable isolation level.
  Kevin Grittner and Dan Ports
  Tue Feb 8 2011, Postgres 9.1
- First SERIALIZABLE implementation using serializable snapshot isolation (best in class)
- SLRU buffer of 16 pages
  - ... but lookups only occur once per command in serializable transactions
  - Much lower frequency
  - Each item is 8 bytes long

Development History 8: Make pg_clog size adaptive

- Commit 33aaa139e630:
  Make the number of CLOG buffers adaptive, based on shared_buffers.
  Robert Haas
  Fri Jan 6 2012, Postgres 9.2
- First case of runtime-determined SLRU size
  - 32 buffers with shared_buffers = 128 MB and up
- But not directly configurable!

Development History: 9. pg_commit_ts

- Commit Timestamps
- Commit 73c986adde5d:
  Keep track of transaction commit timestamps
  Álvaro Herrera
  Wed Dec 3 2014, Postgres 9.5
- 12 bytes per entry
- For use with BDR
  - open-source bi-directional replication implementation
  - uses timestamp for simplistic conflict resolution
- Size is adaptive like `pg_clog`, but grows more slowly and the upper limit is smaller (16 buffers)
- (Theory behind this: not needed for long)

What SLRUs exist

- pg_commit (neé pg_clog), adaptive 8-32 pages
- pg_subtrans, 32 pages
- pg_multixact/offset, 8 pages
- pg_multixact/members, 16 pages
- pg_notify, 8 pages
- pg_serial, 8 pages
- pg_commit_ts, adaptive 8-16 pages
- Total memory use: 88 pages = between 704 kB and 960 kB

Developments for PostgreSQL 17

Borodin: Performance Problem Reported

Andrey Borodin reports to pgsql-hackers:

> *I'm investigating some cases of reduced database performance due to MultiXactOffsetLock contention (80 % MultiXactOffsetLock, 20 % IO DataFileRead). The problem manifested itself during index repack and constraint validation. Both being effectively full table scans.*

pgsql-hackers: MultiXact\SLRU buffers configuration

(Fri, 8 May 2020)

Borodin: Performance Problem Reported (2)

```
database=# SELECT pid, wait_event, wait_event_type, state, query
database-# FROM pg_stat_activity    \watch 1
  pid  |         wait_event          | wait_event_type | state  |
-------+----------------------------+-----------------+--------+----------------
 41344 | ClientRead                 | Client          | idle   | insert into t1
 41375 | MultiXactOffsetControlLock | LWLock          | active | select * from
 41377 | MultiXactOffsetControlLock | LWLock          | active | select * from
 41378 |                            |                 | active | select * from
 41379 | MultiXactOffsetControlLock | LWLock          | active | select * from
 41381 |                            |                 | active | select * from
 41383 | MultiXactOffsetControlLock | LWLock          | active | select * from
 41385 | MultiXactOffsetControlLock | LWLock          | active | select * from
(8 rows)
```

Borodin: Performance Problem Reported (3)

Andrey Borodin:

> *I've found out that:*
>
> *1. When MultiXact working set does not fit into buffers - benchmark results grow very high. Yet, very big buffers slow down benchmark too. For this benchmark optimal SLRU size [is] 32 pages for offsets and 64 pages for members (defaults are 8 and 16 respectively).*

Darold: Patch Benchmarking Results

Gilles Darold:

> *Some time ago I have encountered a contention on MultiXactOffsetControlLock with a performance benchmark. Here are the wait event monitoring result with a polling each 10 seconds and a 30 minutes run for the benchmark:*

```
 event_type |           event           |   sum
------------+---------------------------+----------
 Client     | ClientRead                | 44722952
 LWLock     | MultiXactOffsetControlLock | 30343060
 LWLock     | multixact_offset          | 16735250
 LWLock     | MultiXactMemberControlLock |  1601470
 LWLock     | buffer_content            |   991344
```

Darold: Patch Benchmarking Results (2)

Gilles Darold:

> *After reading this thread I changed the value of the buffer size to 32 and 64 and obtain the following results:*

```
 event_type |            event             |    sum
------------+------------------------------+-----------
 Client     | ClientRead                   | 268297572
 LWLock     | MultiXactMemberControlLock   |  65162906
 LWLock     | multixact_member             |  33397714
 LWLock     | buffer_content               |   4737065
```

Darold: Patch Benchmarking Results (3)

Gilles Darold:

> *I have increased the buffers to 128 and 512 and obtain the best results for this benchmark:*
> *Increasing buffer sizes to (128, 512)*

```
 event_type |            event             |    sum
------------+------------------------------+-----------
 Client     | ClientRead                   | 160463037
 LWLock     | MultiXactMemberControlLock |   5334188
 LWLock     | buffer_content               |   5228256
 LWLock     | buffer_mapping               |   2368505
 LWLock     | SubtransControlLock          |   2289977
```

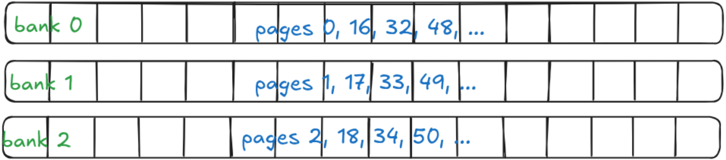
Increasing Buffer Sizes is not Enough

Andrey Borodin again:

> *I have one more idea inspired by CPU caches. Let's make SLRU n-associative, where n ~ 8. We can divide buffers into "banks", number of banks must be power of 2. All banks are of equal size. We choose bank size to approximately satisfy user's configured buffer size. Each page can live only within one bank. We use same search and eviction algorithms as we used in SLRU, but we only need to search/evict over 8 elements.*

- pgsql-hackers: MultiXact\SLRU buffers configuration ↗ (Sun, 11 Apr 2021)
- Dividing the buffers in banks allows much larger buffer sizes
- ... without affecting performance of buffer search

SLRU banks


bank 0
pages 0, 16, 32, 48, ...
bank 1
pages 1, 17, 33, 49, ...
bank 2
pages 2, 18, 34, 50, ...

New Kid in Town: pg_stat_slru

- pg_stat_slru was born as the initial problem was being discussed
- Commit 28cac71bd368: ↗
  Collect statistics about SLRU caches
  Tomas Vondra

  Thu Apr 2 2020, Postgres 13
- No motivation for this work was admitted other than healthy scientific curiosity

Example pg_stat_slru

```
SELECT name, blks_zeroed, blks_hit, blks_read, blks_written
  FROM pg_stat_slru;
```

name	blks_zeroed	blks_hit	blks_read	blks_written
commit_timestamp	1284048	387594150	54530	1305858
multixact_member	30252	23852620477	48555852	26106
multixact_offset	10637	23865848376	18434993	9375
notify	0	0	0	0
serializable	0	0	0	0
subtransaction	513486	12127027243	153119082	431238
transaction	32107	22450403108	72043892	18064
other	0	0	0	0

(“name” column differs in versions prior to 17)

Finalizing a Solution

- Dilip Kumar further analyzed the problem ↗ as reported by customers, created additional pgbench reproducer workloads and posted a new proposal:

  Just increasing the size of the buffer pool doesn't necessarily help, because the linear search that we use for buffer replacement doesn't scale ...

- (We know! Which is why the patch uses banks)

  ... and also because contention on the single centralized lock limits its scalability.

Proposed Changes to SLRUs

In addition to Andrey Borodin's ideas:

- Configurable buffer sizes
- Split each SLRU area in banks

Dilip Kumar proposed:

- Modify operations to LRU counter to use atomic access
- Make the locking occur per bank rather than globally

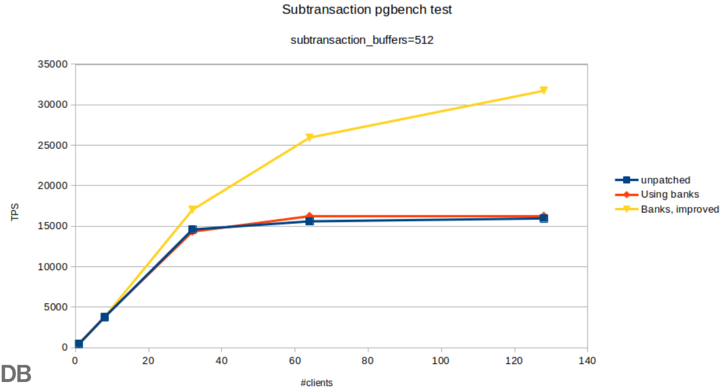
Operating with atomics

1. Scan linearly the array of buffers to see if one contains the page we want
2. If we find it, we're done
3. If not, the scan has chosen a "victim" buffer to evict (least recently used)
    1. Evict it, leaving buffer free
    2. Load our page onto our buffer
    3. Increment "recently used" counter using atomic ops
4. Now the page in buffer can be processed

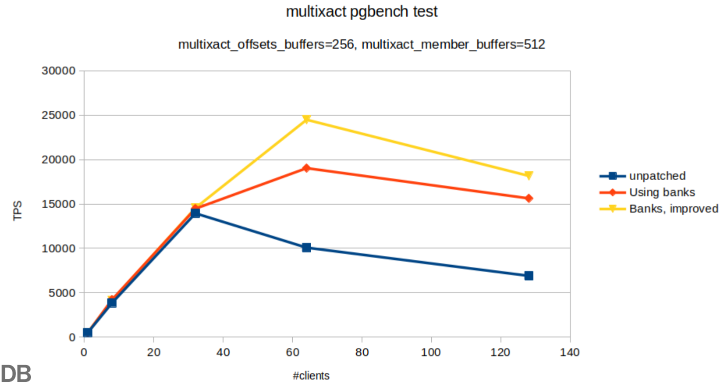
Restricting the scan to banks

1. Scan linearly the array of buffers of the bank that contains the page to see if one contains the page we want
2. If we find it, we're done
3. If not, the scan has chosen a "victim" buffer to evict (least recently used)
    1. Evict it, leaving buffer free
    2. Load our page onto our buffer
    3. Increment "recently used" counter using atomic ops
4. Now the page in buffer can be processed

Subtransaction TPS improvement



Multixact TPS improvement



Performance Fixes in Postgres 17

- Commit d172b717c6f4:
  Use atomic access for SlruShared->latest_page_number
  Dilip Kumar (via Álvaro Herrera)

  Tue Feb 6 2024, Postgres 17

- Commit 53c2a97a9266:
  Improve performance of subsystems on top of SLRU
  Andrey Borodin and Dilip Kumar (via Álvaro Herrera)

  Wed Feb 28 2024, Postgres 17

The New GUCs

- A few must be set to nonzero values, defaults similar as before
- Up to 1024 MB in multiples of 16 (the bank size)

```
# SLRU buffers (change requires restart)
multixact_offset_buffers = 16
multixact_member_buffers = 32
notify_buffers = 16
serializable_buffers = 32
```

The New GUCs: autoscaling

- A few are automatically derived from `shared_buffers`

```
# SLRU buffers (change requires restart)
commit_timestamp_buffers = 0
subtransaction_buffers = 0
transaction_buffers = 0
```

- 2 MB for each 1024 MB of shared_buffers
- Up to a maximum of 8 MB
- Can still be set manually

Action Items for Postgres DBAs

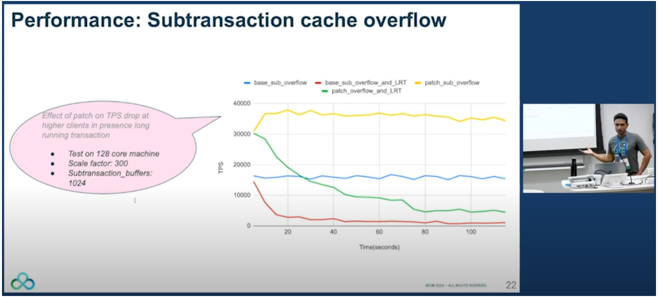
- Add `pg_stat_slru` to monitoring
- Upgrade to Postgres 17
- Track whether any of the SLRUs need to be reconfigured

Suggested Monitoring Query

```
SELECT name, blks_zeroed, blks_read,
     blks_hit+blks_read AS blks_accessed,
     CASE WHEN blks_hit+blks_read = 0 THEN 'NaN'
     ELSE (blks_hit::numeric / (blks_hit+blks_read))
          ::numeric(4,2) END AS hit_ratio
  FROM pg_stat_slru;
```

Another Talk on this Topic

“SLRU Performance Issues: How we have optimized it”, Dilip Kumar, PGConf.dev 2024



Performance: Subtransaction cache overflow

Effect of patch on TPS drop at higher clients in presence long running transaction

- Test on 128 core machine
- Scale factor: 300
- Subtransaction_buffers: 1024

Future Work

- Use per-bank LRU counters
    - avoids latency of atomic ops
- Improve LRU algorithm
    - Ants Aasma has said he has better code; waiting for him to post it
    - I'm especially hoping it'll avoid my stupid use of integer division

Thanks!

Questions?

Álvaro Herrera, EDB
EDB (geb. EnterpriseDB)

alvherre@kurilemu.de
Mastodon: https://lile.cl/@alvherre/