

What is an SLRU anyway?

Álvaro Herrera – PostgreSQL developer, EDB



PostgreSQL Prague Developer Day
Prague, Czechia
29th January 2025

What is an SLRU?

- **S**imple **L**east **R**ecently **U**sed
- A mechanism to store transactional metadata
 - And things with similar behavior
- Metadata examples:
 - Transaction commit/abort status
 - LISTEN / NOTIFY data
 - transaction commit times
- Keeps fixed-size memory buffer of on-disk data

Development History: 1. pg_clog

- Simplistic pg_log pseudo-relation 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 18:52:43 2001 +0000, Postgres 7.2
- Initially, LRU is an internal pg_clog implementation detail
- Hardcoded buffer size of 8 pages
 - Much later, pg_clog was renamed pg_xact, commit 88e66d193fba (2017).

Development History: 1. pg_clog

- Simplistic pg_log pseudo-relation 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 18:52:43 2001 +0000, Postgres 7.2
- Initially, LRU is an internal pg_clog implementation detail
- Hardcoded buffer size of 8 pages
 - Much later, pg_clog was renamed pg_xact, commit 88e66d193fba (2017).

How does pg_clog work

- Two bits per transaction:
 - 00 → “in-progress”
 - 01 → “aborted”
 - 10 → “committed”
- 0x100000 (decimal 1048576) transactions per file
- Four transactions per byte, 32768 transactions in one 8kB page
- 32 pages per file
- Files whose pages are all old enough can be removed
- 8 in-memory pages store the status of $32768 * 8 = 262144$ transactions

How does pg_clog work (2)

- Reading a tuple requires looking up status of its creating and deleting transactions
- If committed, the result of this lookup is written in the tuple metadata (“hint bits” in the “infomask”)
- Eventually, all older tuples are hinted and no more lookups are needed
- As long as older tuples are “hinted” within 256k transactions, little disk access is needed for pg_clog

Development History: 2. slru.c

- slru.c was born for *nested transactions* from pg_clog shortly thereafter
- Commit 0abe7431c6d7: [↗](#)
This patch extracts page buffer pooling and the simple least-recently-used strategy from clog.c into slru.c.
Bruce Momjian for Manfred Koizar, Wed Jun 11 22:37:46 2003 +0000,
Postgres 7.4
- The term “slru” was invented at this point
- Nobody thought this name would ever be exposed to users

Development History: 3. pg_subtrans (2)

- pg_subtrans stores the transaction ID of the parent of each transaction
- Commit 573a71a5da70: [↗](#)
Nested transactions.
Tom Lane for Álvaro Herrera,
Thu Jul 1 00:52:04 2004 +0000, Postgres 8.0
- First user of slru.c outside pg_clog
- 4 bytes per transaction (16x larger than pg_clog!)
- 8 pages of 8kB each have room for 16536 transactions

How does pg_subtrans work?

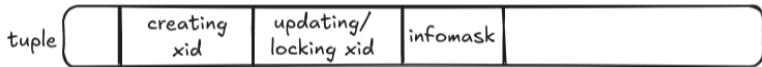
- pg_subtrans responds to “is transaction X running?” in presence of subtransactions
- ... but only for transactions with >64 subtransactions
- therefore, access is rare.

- With no subtransactions, shared memory access is sufficient to know if a transaction is running
- We keep a cache of 64 running subtransactions in shared memory
- Accessing pg_subtrans is only needed if the cache has “overflowed”

Development History: 4. pg_multixact

- Commit bedb78d386a4: [↗](#)
Implement sharable row-level locks, and use them for foreign key references to eliminate unnecessary deadlocks.
Álvaro Herrera and Tom Lane
Thu Apr 28 21:47:18 2005 +0000, Postgres 8.1
- A two-level mechanism to store variable-sized arrays for a single lookup key:
 - 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

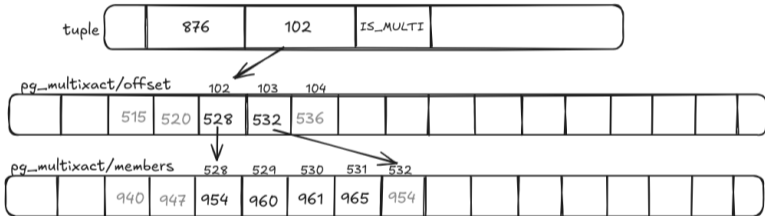
How does pg_multixact work?



How does pg_multixact work?



How does pg_multixact work?



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 23:08:34 2005 +0000, 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 23:08:34 2005 +0000, 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 23:08:34 2005 +0000, Postgres 8.2

(Short) theory of operation

When we require status of a transaction:

- 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)
- 4 Evict it, leaving buffer free
- 5 Load our page onto our buffer
- 6 Increment “recently used” counter
- 7 Now we can read the data we wanted

Development History: 6. pg_notify

- Commit d1e027221d02: [↗](#)
Replace the pg_listener-based LISTEN/NOTIFY mechanism with an in-memory queue.
Tom Lane for Joachim Wieland
Tue Feb 16 22:34:57 2010 +0000, 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

- pg_serial
- Commit dafaa3efb75ce: [↗](#)
Implement genuine serializable isolation level.
Heikki Linnakangas for Kevin Grittner and Dan Ports
Tue Feb 8 00:09:08 2011 +0200, Postgres 9.1
- First SERIALIZABLE implementation using *serializable snapshot isolation* (best of 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: 7. pg_serial

- pg_serial
- Commit dafaa3efb75ce: [↗](#)
Implement genuine serializable isolation level.
Heikki Linnakangas for Kevin Grittner and Dan Ports
Tue Feb 8 00:09:08 2011 +0200, Postgres 9.1
- First SERIALIZABLE implementation using *serializable snapshot isolation* (best of 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 14:30:23 2012 -0500, Postgres 9.2
- First case of runtime-determined SLRU size
 - 32 buffers with `shared_buffers=128MB` and up
- But not directly configurable!

Development History: 9. `pg_commit_ts`

- `pg_commit_ts`: commit timestamps
- Commit 73c986adde5d: [↗](#)
Keep track of transaction commit timestamps
Álvaro Herrera, Wed Dec 3 11:53:02 2014 -0300, Postgres 9.5
- 12 bytes per entry
- For use with BDR
 - open-source bi-directional replication implementation
 - ... for 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_xact (néé pg_clog), adaptive
- 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

Performance Problem Reported (1)

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 21:36:40 +0500)

- Using artificial reproducer

```
database=# SELECT pid, wait_event, wait_event_type, state, query
database=# FROM pg_stat_activity \watch 1
```

Friday, 8 Mar 2020 15:08:37 (every 1s)

pid	wait_event	wait_event_type	state	query
41344	ClientRead	Client	idle	insert into t1 select generate_series(1,
41375	MultiXactOffsetControlLock	LWLock	active	select * from t1 where i = ANY (\$1) for s
41377	MultiXactOffsetControlLock	LWLock	active	select * from t1 where i = ANY (\$1) for s
41378			active	select * from t1 where i = ANY (\$1) for s
41379	MultiXactOffsetControlLock	LWLock	active	select * from t1 where i = ANY (\$1) for s
41381			active	select * from t1 where i = ANY (\$1) for s
41383	MultiXactOffsetControlLock	LWLock	active	select * from t1 where i = ANY (\$1) for s
41385	MultiXactOffsetControlLock	LWLock	active	select * from t1 where i = ANY (\$1) for s


Performance Problem Reported (2)

```
CREATE TABLE eventwaits (  
    tstamp timestamp with time zone,  
    count int,  
    event_type text,  
    event text  
);  
INSERT INTO eventwaits  
    SELECT now(), count(*), wait_event_type, wait_event  
        FROM pg_stat_activity  
        WHERE state = 'active' AND  
            wait_event_type NOT IN ('Timeout', 'Client', 'Activity')  
    \watch 0.01
```

How to detect a problem

- ... or use `pg_wait_sampling`
- https://github.com/postgrespro/pg_wait_sampling

History of the proposed fix

- [pgsql-hackers: MultiXact\SLRU buffers configuration](#) 
(Fri, 8 May 2020 21:36:40 +0500)
- Andrey Borodin proposes configurable buffer sizes in postgresql.conf

Performance Problem Reported (2)

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

What was the Performance Problem (2)

Gilles Darold:

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

Increasing buffer sizes from (8, 16) to (32, 64):

event_type	event	sum
Client	ClientRead	268297572
LWLock	MultiXactMemberControlLock	65162906
LWLock	multixact_member	33397714
LWLock	buffer_content	4737065

What was the Performance Problem (2)

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

(Short) theory of operation (2)

When choosing a victim buffer:

- 1 Scan all buffers in the array
- 2 If one is marked free, choose that one; we're done
- 3 Keep track of the one with lowest “recently used” counter
- 4 If we scanned all buffers, the victim is the one we memoized

Therefore, a very large array of SLRU buffers is undesirable because scanning it would take a long time

Increasing Buffer Size is not Enough

Andrey Borodin again:

I have one more idea inspired by CPU caches. Let's make SLRU n -associative, where $n \sim 8$. We can divide buffers into "banks", number of banks must be power of 2. [...] 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 21:37:21 +0300)
- Dividing the buffers in *banks* allows much larger buffer sizes
- ... without affecting performance of buffer search

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 02:11:38 2020 +0200, Postgres 13

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	10638	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

Monitoring SLRU cache ratios

suggested monitoring

```
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;
```

Monitoring numbers

```
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;
```

<i>name</i>	<i>blks_zeroed</i>	<i>blks_read</i>	<i>blks_accessed</i>	<i>hit_ratio</i>
commit_timestamp	2674	1	2148271	1.00
multixact_member	158	257	309927	1.00
multixact_offset	63	117	309630	1.00
notify	0	0	0	NaN
serializable	0	0	0	NaN
subtransaction	2673	0	390133	1.00
transaction	166	796	20609643	1.00
other	0	0	0	NaN

Finalizing a Solution

Dilip Kumar further analyzed the problem on customer systems, created reproducers 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, and also because contention on the single centralized lock limits scalability.

pgsql-hackers: SLRU optimization - configurable buffer pool and partitioning the SLRU lock (Wed, 11 Oct 2023 16:34:37 +0530) [↗](#)

Proposed Changes to SLRUs

In addition to Andrey Borodin's ideas:

- Configurable buffer sizes
- Split each buffer area in banks

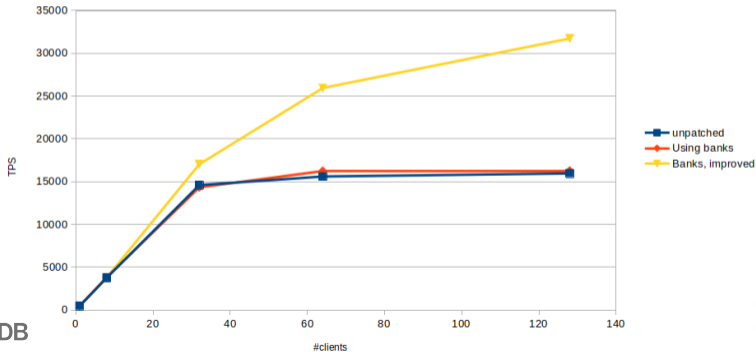
Dilip Kumar proposed:

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

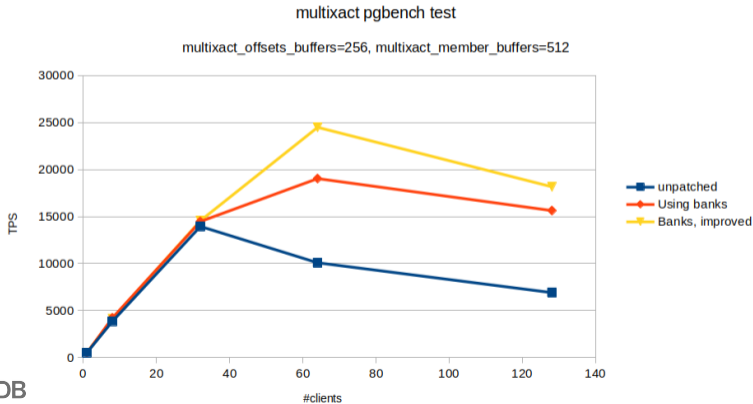
Subtransaction TPS improvement

Subtransaction pgbench test

subtransaction_buffers=512



Multixact TPS improvement



Performance fixes in Postgres 17

- Commit d172b717c6f4: [↗](#)
Use atomic access for SlruShared->latest_page_number
Álvaro Herrera for Dilip Kumar,
Tue Feb 6 10:54:10 2024 +0100, Postgres 17
- Commit 53c2a97a9266: [↗](#)
Improve performance of subsystems on top of SLRU
Álvaro Herrera for Andrey Borodin and Dilip Kumar,
Wed Feb 28 17:05:31 2024 +0100, Postgres 17

The new GUCs

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

new postgresql.conf lines, defaults

```
# 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 on shared_buffers:

new postgresql.conf lines

```
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

Thanks!

Questions?

Álvaro Herrera, EDB

`alvherre@alvh.no-ip.org`

`alvaro.herrera@enterprisedb.com`

Mastodon: <https://lile.cl/@alvherre/>