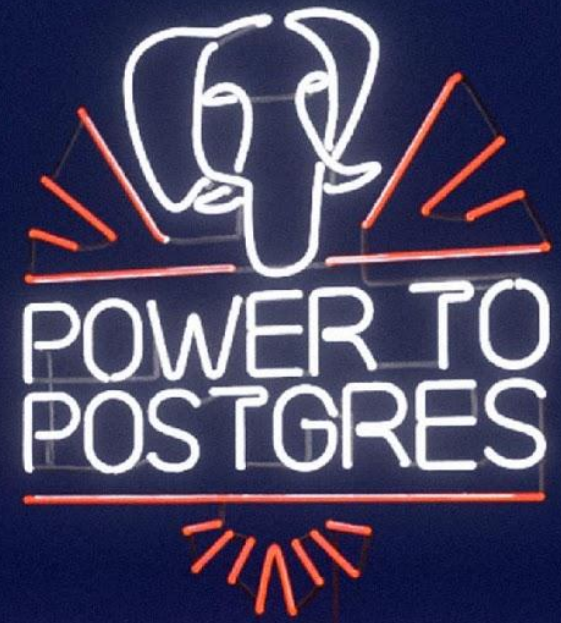
Rare but extremely challenging PostgreSQL Performance Problems

Dilip Kumar

24-Feb-2023 @PGConf India



EDB

Agenda:

- Problem introduction - 1
- Snapshot
- PG_SUBTRANS SLRU
- Visibility check
- Subtransaction cache overflow
- Presence of long running transactions
- Problem introduction - 2
- Row locking
- Multixact
- Summary
- Q&A

EDB™

Problem Introduction - 1

SETUP

scale factor: 300
shared_buffers=8GB
checkpoint_timeout=40min
max_wal_size=20GB
max_connections=200
maintenance_work_mem=1GB&
./pgbench -c 64 -j 64 -T 1800 -P5 -M prepared postgres

86595	LWLock	SubtransSLRU
14619	LWLock	SubtransBuffer
2574	Client	ClientRead
1876	Activity	WalWriterMain
1799	Lock	transactionid
1717	Timeout	PgSleep
1093	Activity	BgWriterMain
404	IO	SLRURead

RESULT

progress: 5.0 s, 30008.0 tps, lat 2.117 ms stddev 1.390, 0 failed

progress: 10.0 s, 30134.3 tps, lat 2.123 ms stddev 1.280, 0 failed

progress: 530.0 s, 28612.6 tps, lat 2.236 ms stddev 1.272, 0 failed

——-> create subtransaction overflow <——----------

progress: 545.0 s, 15876.0 tps, lat 4.011 ms stddev 2.604, 0 failed

progress: 705.0 s, 16366.2 tps, lat 3.909 ms stddev 3.292, 0 failed

——-> start a long running idle transaction<——----------

progress: 710.0 s, 15762.8 tps, lat 4.060 ms stddev 3.236, 0 failed

progress: 715.0 s, 5604.2 tps, lat 11.399 ms stddev 6.955, 0 failed

….

progress: 1790.0 s, 227.4 tps, lat 268.356 ms stddev 368.967, 0 failed

progress: 1795.0 s, 264.6 tps, lat 250.147 ms stddev 359.489, 0 failed

EDB

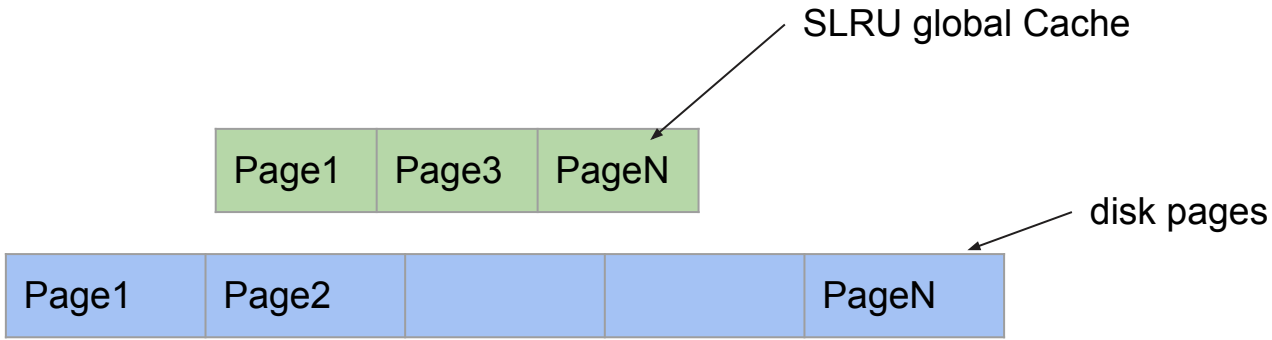
Snapshot

- We need a snapshot for getting a consistent view of the data (ACID)
- All running (sub)transaction ids are kept in snapshot's xip/subxip array
- Snapshot is bounded with xmin and xmax
- Snapshot is prepared by reading xid and subxid information from each backend
- Each backend can cache 64 subxids

EDB

PG_SUBTRANS SLRU

- PG_SUBTRANS maintains the parent xid of each subxid
- Need to update this information whenever we assign a subtransaction id
- Reader do not need to access this SLRU in normal cases
  - Snapshot should contains subtransaction information in subxip array



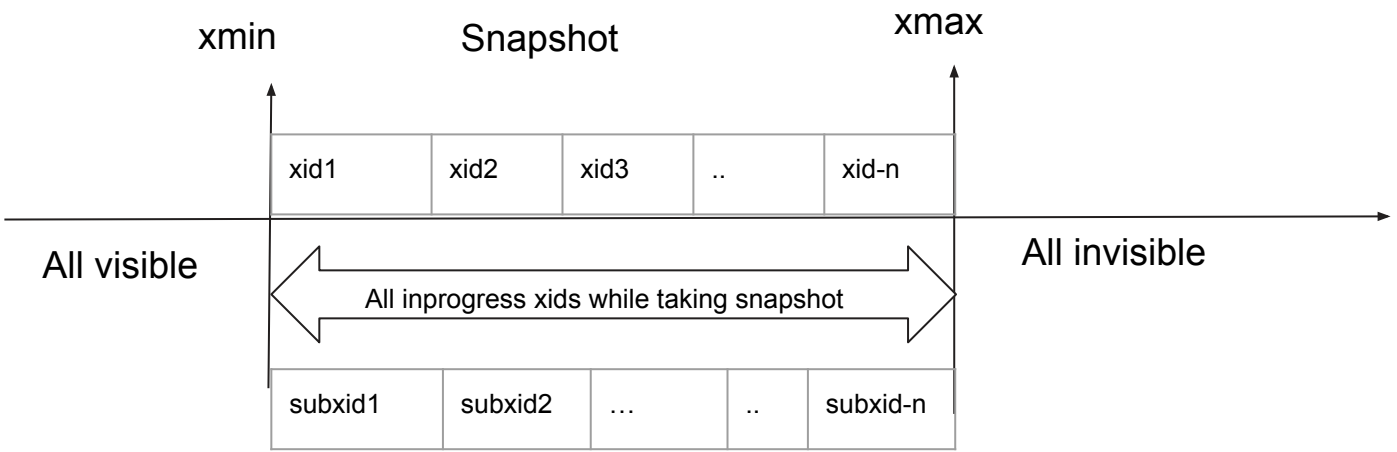
Visibility check

- Each tuple has xmin and xmax
  - it has different meaning than snapshot xmin and xmax

- Tuple xids (xmin/xmax) falling within the snapshot xmin/xmax range need to be looked into the snapshot’s subxip and xip array

EDB

Visibility check



Xmin and xmax in Heap Page

xmin	xmax	tuple1
xmin	xmax	tuple2
xmin	xmax	tuple3

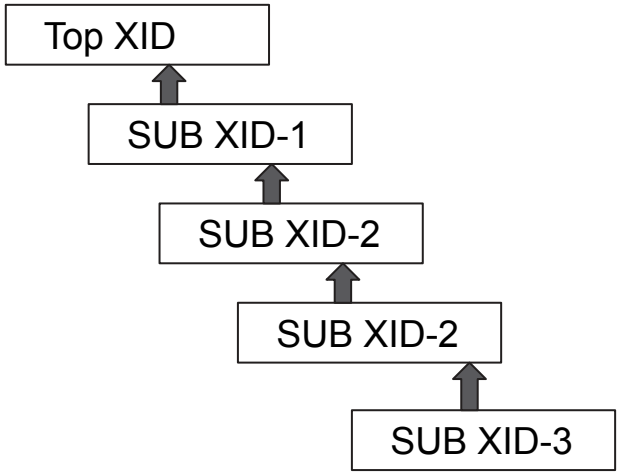
Subtransaction cache overflow

- Per backend there is cache to hold 64 subtransaction
  - If this limit cross we no longer maintain the subxid informations
  - Now snapshot->subxip is not complete
    - We can only rely on snapshot->xip array but not on snapshot->subxip array
    - We need to access the pg_subtrans SLRU to get parent xid and then look into xip array
- Problems in accessing pg_subtrans SLRU
  - This is accessed under LWLock
  - Reader and Writer conflict on Lock
  - Need to access disk if pages are not in SLRU buffer

EDB

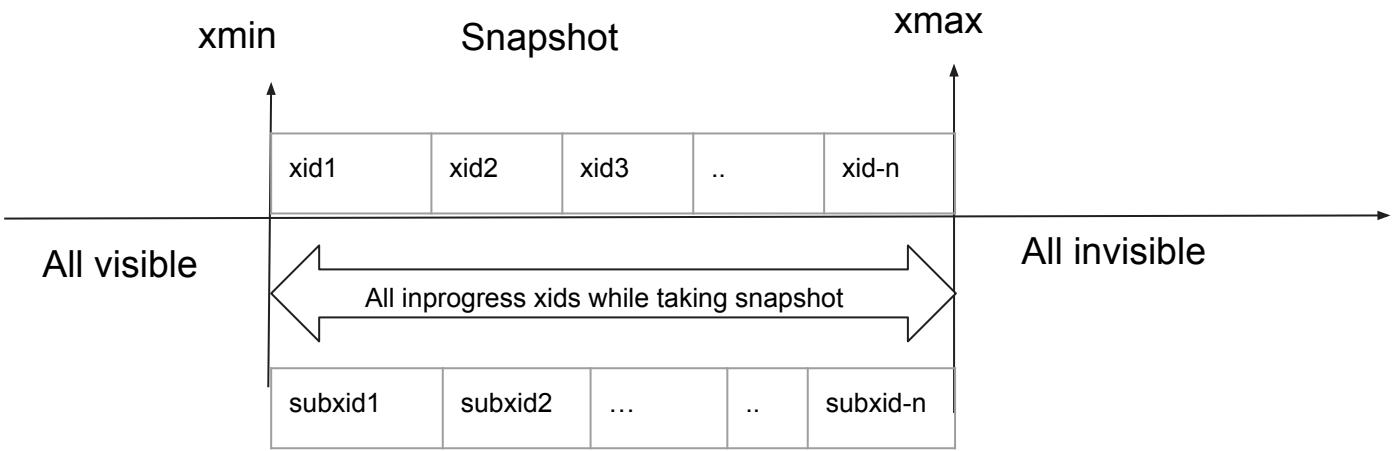
Subtransaction cache overflow

- Need to traverse the subtransaction tree in the pg_subtrans slru to get the top transaction id



Presence of long running transaction

- What is long running transaction
  - Assign xid and do not commit
- Impact on visibility
  - Snapshot has large range of xmin and xmax
    - More access to SLRU
  - Frequent SLRU cache miss



Release Savepoint vs Rollback To Savepoint

- Rollback to Savepoint
  - Revert the effect of partial transaction
  - Subxid is immediately marked aborted in PG_XACT
  - Now we can release the subxid slot from backend cache

- Release savepoint
  - Release the current active subtransaction from stack
    - Help reducing the depth of subxact tree
  - Subxid can not be considered committed until top xid is committed
  - Can not release the slot from the session cache

EDB

Subtransaction cache overflow - Summary

- Frequent subtransaction overflow can create slow down due to SLRU lookup
- This become worst in presence of long running transaction(s)
  - Because that will force more xids to be looked up into the SLRU
  - Create contention on SLRU lock as well as frequent SLRU buffer replacement
- Use release savepoint whenever possible to reduce the subxact tree depth

EDB

Problem Introduction - 2

```
BEGIN;
SELECT FROM pgbench_accounts WHERE aid = :aid FOR
UPDATE;
SAVEPOINT S1;
UPDATE pgbench_accounts SET abalance = abalance + :delta
WHERE aid = :aid;
SELECT abalance FROM pgbench_accounts WHERE aid =
:aid;
END;
```

```
16729  LWLock     | MultiXactOffsetSLRU
 7086  LWLock     | MultiXactOffsetBuffer
 1313  Client     | ClientRead
  443  Activity   | LogicalLauncherMain
  443  Activity   | CheckpointerMain
  443  Activity   | AutoVacuumMain
```

RESULT

```
progress: 5.0 s, 32610.6 tps, lat 1.949 ms stddev 0.612, 0 failed
progress: 10.0 s, 33412.6 tps, lat 1.915 ms stddev 0.625, 0 failed
progress: 15.0 s, 32396.0 tps, lat 1.974 ms stddev 0.745, 0 failed
.....
progress: 20.0 s, 32109.6 tps, lat 1.993 ms stddev 0.799, 0 failed
—--> start a long running idle transaction←—----------

progress: 100.0 s, 26762.9 tps, lat 2.390 ms stddev 2.685, 0 failed
progress: 105.0 s, 17583.5 tps, lat 3.636 ms stddev 2.907, 0 failed
progress: 110.0 s, 7990.4 tps, lat 7.993 ms stddev 9.713, 0 failed
progress: 115.0 s, 4305.4 tps, lat 14.837 ms stddev 20.893, 0 failed
....

progress: 745.0 s, 1375.4 tps, lat 46.531 ms stddev 22.377, 0 failed
progress: 750.0 s, 1359.8 tps, lat 46.982 ms stddev 22.115, 0 failed
progress: 755.0 s, 1352.2 tps, lat 47.423 ms stddev 22.168, 0 failed
```

EDB

Row Locking

- A regular select statement does not give you enough protection if you want to query data and make a change in the database related to it.
  - Other transactions can update or delete the data you just queried.
- With ROW Locking PostgreSQL offers additional select statements that lock on read and provide an extra layer of safety.
  - The *select for update* - acquire a exclusive lock on row(s)
    - Mainly require for selecting first and updating later
  - The *select for share* acquires a shared lock
    - Can lock a particular row in primary key table before inserting in foreign key table

EDB

Row Locking

- PostgreSQL maintains the xid of an inserter in xmin and xid of an deleter/updater in xmax field of the tuple
- Xid of the locker is also maintained in the xmax field
- Now we are allowed to acquire multiple concurrent row shared lock
  - So how to store multiple xid in one xmax field?

MultiXact

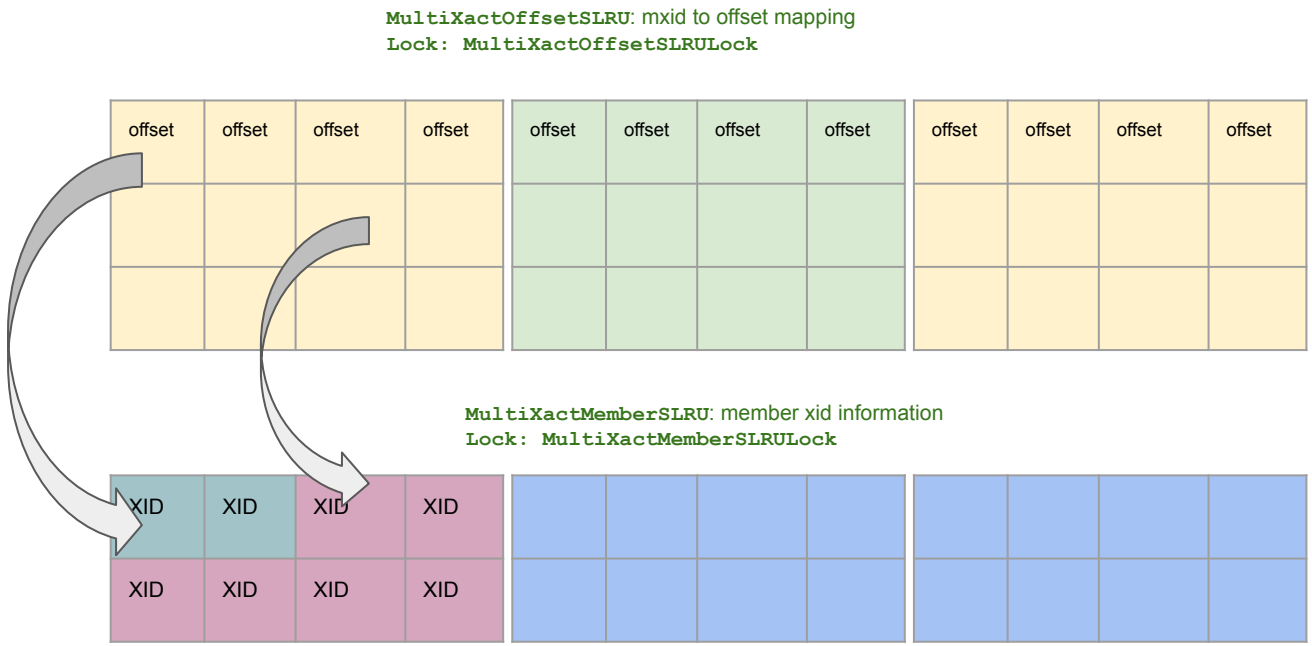
- Multixact provide a way to create a new kind of xid called multi-xact id
  - Under the hood it is a mapping from a new xid called multi-xact id to a list of xids and their locking status

- The pg_multixact manager stores an array of MultiXactMember for each MultiXactId.

MultiXact

- Number of member xid for the multixact-id is variable so we can not map to a fixed offset
- For handling this we use two SLRU
  - One for storing the offsets at which the data starts for each MultiXactId
  - And in other one we store variable length of transaction ids

MultiXact



MultiXact

- When MultiXact ID is created
  - Generally it required a share locker
    - SELECT for SHARE
    - SELECT for NO KEY UPDATE
    - SELECT for KEY SHARE
  - Special case when it is created by single transaction with a subtransaction
    - Special Case

```
BEGIN

SELECT … FOR UPDATE;

SAVEPOINT s1;

UPDATE
```

MultiXact Summary

- Generally share lockers create multixact id
- Long running can block cleanup of hot chains
    - This will create more contention on the MultiXact SLRU

EDB

Q & A

Thank You