

DON'T FEAR LARGE TABLE INDEXING ANYMORE!

Stop worrying about indexing your huge tables!

Two complementary patches — [snapshot resetting](#) and [STIR](#)-based validation dramatically reduce indexing time and let your VACUUM run effectively.

Overview

In PG14, VACUUM learned to ignore CIC snapshots — a major win for long index builds. A concurrency bug forced a **2022 revert**. These two patches (CF #6401 and #4971) bring it back — **safely** — while dramatically improving CIC/RIC speed, advancing the xmin horizon, and reducing vacuum interference. (Original feature: [d9d0762](#); reverted: [e28bb88](#).)

First Phase: Snapshot Reset (CF #6401)

- Snapshots are periodically reset “between” pages to `GetLatestSnapshot()` during the heap scan of the first phase.
- For unique indexes, special handling preserves uniqueness by checking liveness with `SnapshotSelf` during `_bt_load` when equal values arrive in a row.
- Ships with stress tests that reproduce the original 2022 bug and other concurrency issues.

Second Phase: STIR (CF #4971)

The second phase’s job is to add entries inserted into the table during the first phase. Today this needs a full second scan plus index/heap TID comparison.

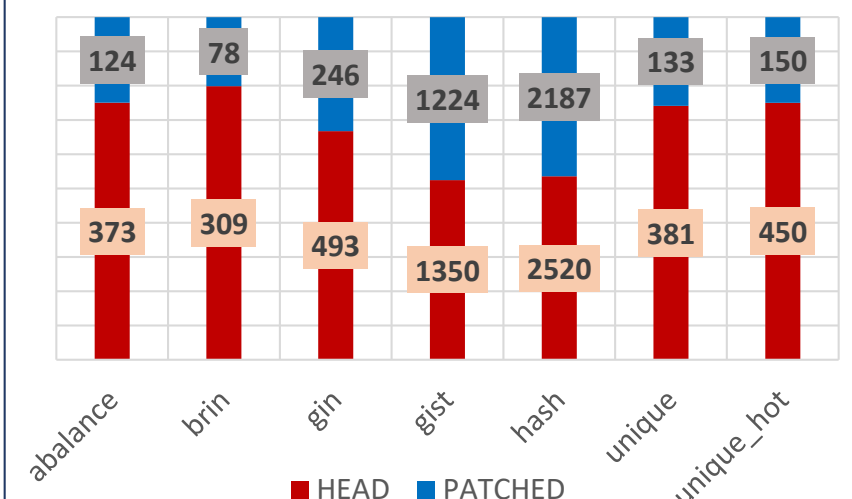
Short Term Index Replacement is a special lightweight auxiliary structure with a single mission – capture every new TID inserted into the heap during the first phase. As a result, we only need to compare index TIDs against TIDs captured by **STIR**! Up to **3x** performance boost!

The **STIR** itself:

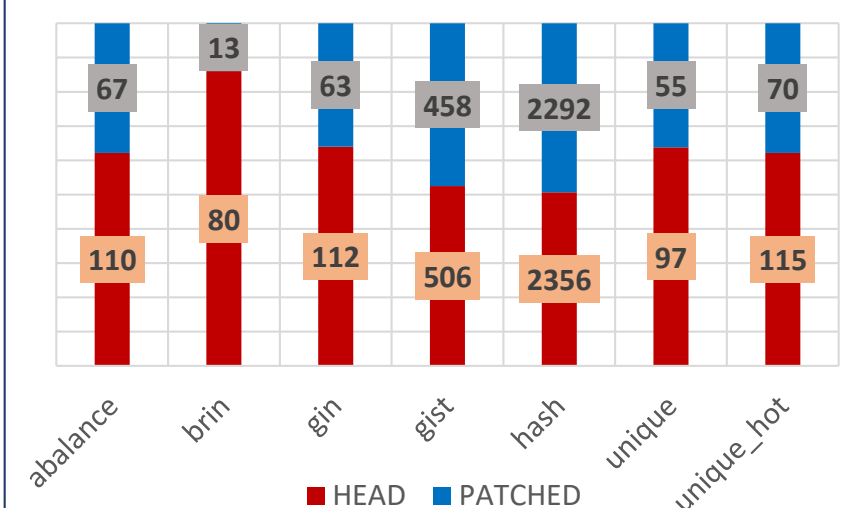
- AM with the same columns, predicates and expressions as a target index
- Always unlogged
- Does not support any queries
- Simply appends new incoming TIDs to its pages
- Since it does not store any indexed data – it is not even prepared during insert
- Automatically dropped if it becomes junk due to errors

Additionally, we are safe to reset snapshots during the new validation phase – we are just operating with root TIDs, so any snapshot is fine! We just need to **WaitForOlderSnapshots** of the newest one before marking the index as valid.

Index build time, disk 1ms delay, less is better



Index build time, local SSD, less is better



Benchmark shows faster index build time for different cases

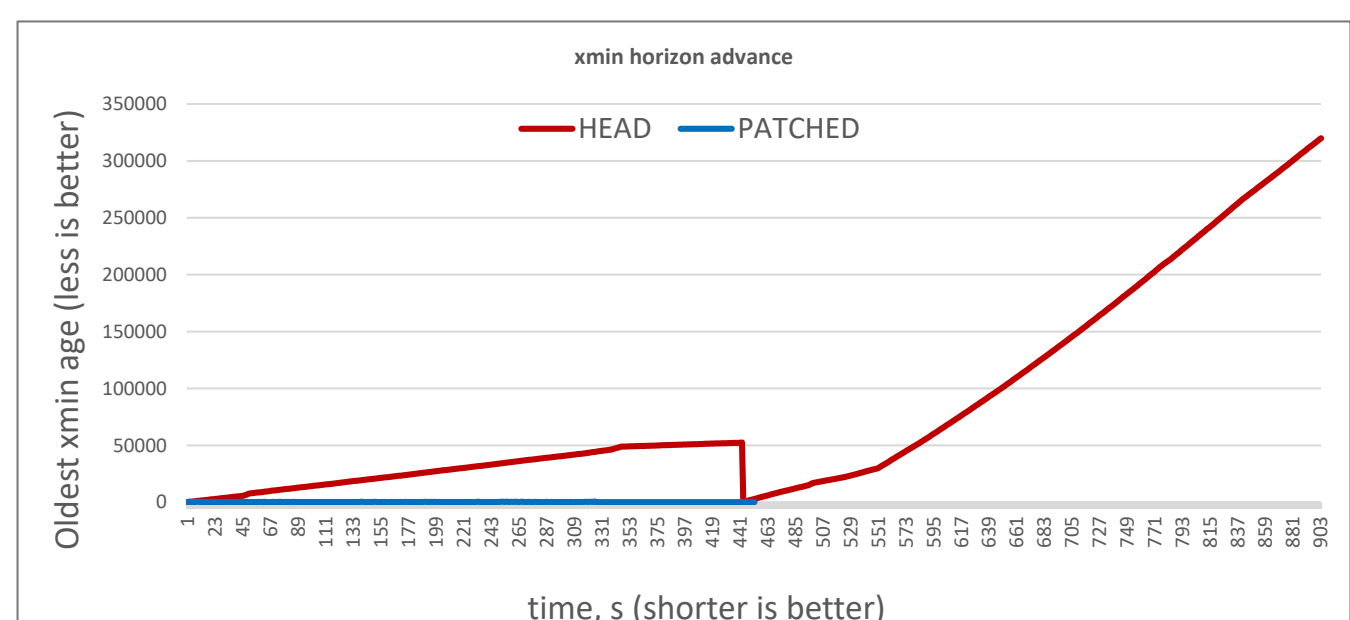
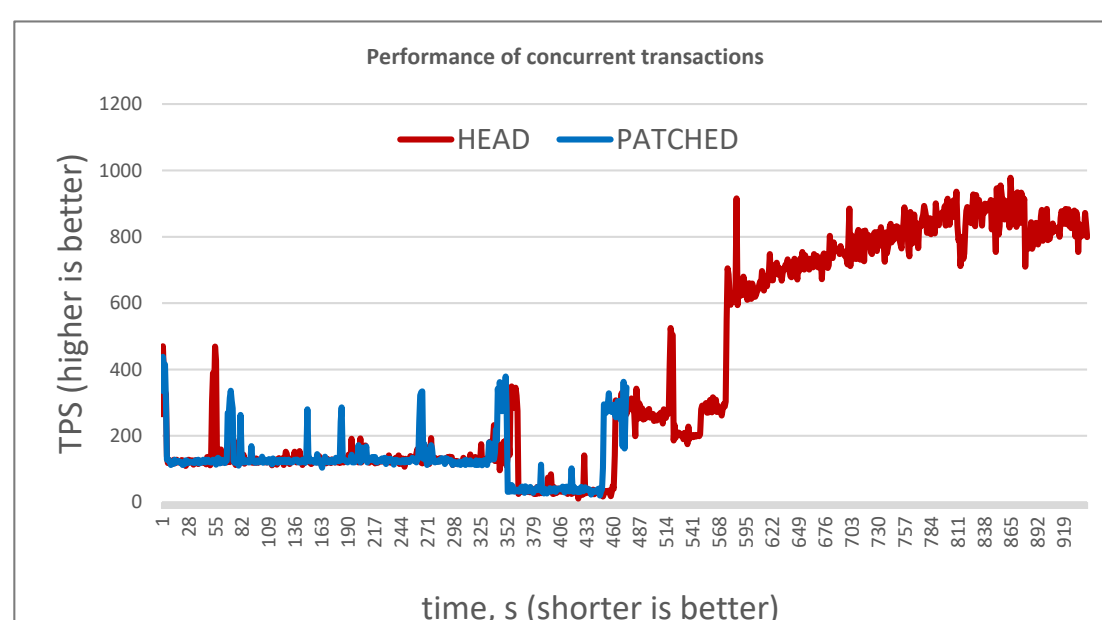
- pgbench scale 2000
- Two types of storage – high-end local SSD or 1ms delay SSD
- 8 concurrent pgbench clients

```
abalance
CREATE INDEX CONCURRENTLY idx ON pgbench_accounts (abalance)
brin
CREATE INDEX CONCURRENTLY idx ON pgbench_accounts USING brin(abalance)
gin
CREATE INDEX CONCURRENTLY idx ON pgbench_accounts using gin(abalance)
gist
CREATE INDEX CONCURRENTLY idx ON pgbench_accounts using gist(abalance)
hash
CREATE INDEX CONCURRENTLY idx ON pgbench_accounts USING hash(bid)
unique
CREATE UNIQUE INDEX CONCURRENTLY idx ON pgbench_accounts (aid)
unique_hot (causes a lot new TIDs coming each update)
CREATE INDEX hot ON pgbench_accounts (abalance)
CREATE UNIQUE INDEX CONCURRENTLY idx ON pgbench_accounts (aid)
```

Benchmark shows 2x faster index build time, same performance of concurrent transactions, effective xmin horizon advance

- pgbench scale 2000
- io2 AWS storage
- 8 concurrent pgbench clients

```
CREATE UNIQUE INDEX
CONCURRENTLY idx ON
pgbench_accounts (aid)
```



Build your indexes confidently, no matter how large the table.

Choose smarter indexing **today**.