DON'T FEAR LARGE TABLE INDEXING ANYMORE!

Stop worrying about indexing huge tables!

Our snapshot-resetting technique and optimized STIR-based validation phase dramatically reduce indexing time and ensure that VACUUM processes run effectively.

- UP TO 3× FASTER CONCURRENT INDEX BUILDS!
- THE XMIN HORIZON ADVANCES THE WHOLE INDEX BUILD DURATION!
- ROBUST AND EFFICIENT—NO SHORTCUTS, BULLETPROOF CLASSIC SNAPSHOTS ONLY!
- STRESS-TESTS INCLUDED!

Overview

This patch dramatically improves concurrent index build (CIC/RIC) speed, while allowing xmin horizon advancement, reducing vacuum interference. The mission is to bring back the PG14 feature introduced in **d9d0762** and reverted in **e28bb88.**

Optimization of First Phase: Snapshot Reset

- Snapshots are periodically reset "between" pages to **GetLatestSnapshot()** during the heap scan of the first phase.
- For unique indexes the special handling ensures uniqueness constraints remain intact by checking liveness with **SnapshotSelf** during **_bt_load** in case of equal values coming in a row.

Optimization of Second Phase: STIR

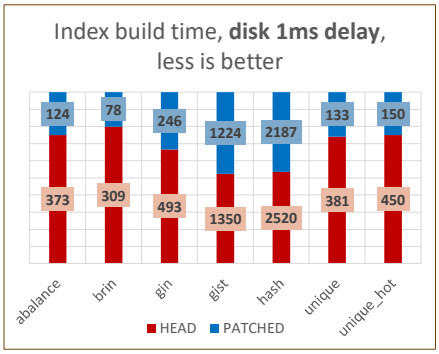
Goal of the second phase is to add entries which were inserted into the table during the first phase. Currently it is done by a second scan of the whole table and comparison of the index TIDs with heap TIDs.

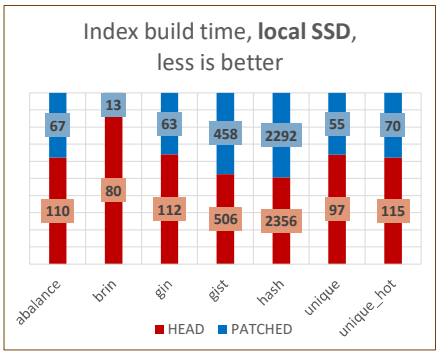
Short **T**erm **I**ndex **R**eplacement is a special lightweight auxiliary structure – it serves a single mission – capture all the new TIDs inserted into the heap during the first phase. As a result, we need to compare index TIDs only with TIDs captured by **STIR**! It gives up to **3×** performance 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 index as valid.





Benchmark shows faster index build time for different cases

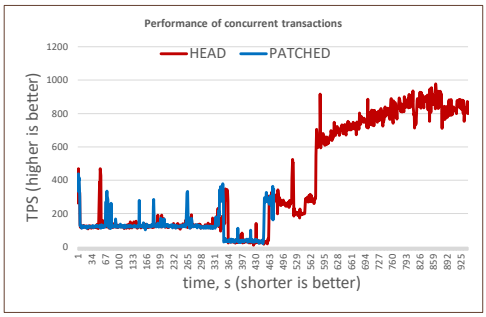
- **pgbench scale 2000**
- **Two types of storage – high-end local SSD or 1ms delay SSD**
- **8 concurrent pgbench clients**

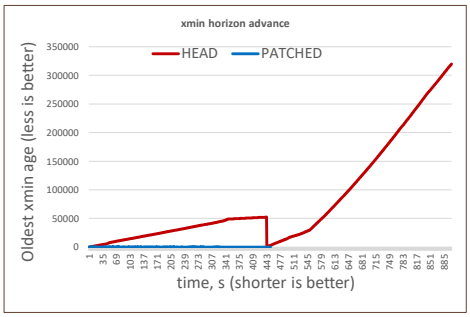
abalance
```
CREATE INDEX CONCURRENTLY idx ON pgbench_accounts (abalance)
```
brin
```
CREATE INDEX CONCURRENTLY idx ON pgbench_accounts USING brin(abalance)
```
gin
```
CREATE INDEX CONCURRENTLY idx ON pgbench_accounts using gin(abalance)
```
gist
```
CREATE INDEX CONCURRENTLY idx ON pgbench_accounts 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 2× 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.

Commitfest item:

4971

Mikhail Nikalayeu

PGConf.dev