
Yandex

Yandex

CREATE INDEX CONCURRENTLY implementation details

Andrey Borodin, Team lead of open source RDBMS development at Yandex.Cloud

About me

Developing PostgreSQL 4 fun and on behalf of Yandex Cloud

- Took part in ~60 patches
- Sometimes contributing to other databases
- Maintain WAL-G, Odyssey, SPQR and some other stuff

Why CIC?

```
% psql -c "begin;
insert into tbl values(1);
select pg_sleep(3);
rollback;" postgres&
% time psql -c "create index on tbl(clmn)" postgres
0.00s user 0.01s system 0% cpu 2.976 total
```

https://www.pgcon.org/2019/schedule/events/1330.en.html

PGCon2019 - 3.4

Index

Tutorials - Day 1: Tuesday - 2019-05-28

Tutorials - Day 2 & Unconference: Wednesday - 2019-05-29

Talks - Day 1 - 2019-05-30

Talks - Day 2 - 2019-05-31

Speakers

PGCon 2019

The PostgreSQL Conference

Challenges of Concurrent DDL

Why is this such a hard problem, and is there anything we can do about it?

Five years after removing SnapshotNow, PostgreSQL still has very little in the way of DDL that can execute concurrently with user queries, and many of the facilities that do exist have significant and sometimes painful limitations. In this talk, I'll give an overview of the problems that have stymied previous attempts to reduce lock levels and design new concurrent DDL facilities.

The removal of SnapshotNow provides only a very limited guarantee about how sane catalog lookups can be assumed to be in the presence of concurrent DDL. The heavyweight lock manager, the shared invalidation system, and the relcache are interrelated subsystems with complex interactions whose basic design presupposes that concurrent DDL does not exist. Query planning relies on certain information remaining static between plan time and execution time both for both efficiency and correctness. Deadlock hazards hinder efforts to keep the time for which strong relation locks are held to a minimum. I'll discuss these and other problems that make implementing concurrent DDL correctly an exceptionally hard problem within the PostgreSQL infrastructure.

Attached files

- (application/pdf - 153.7 KB)

SPEAKERS	
	Robert Haas
SCHEDULE	
Day	Talks - Day 1 - 2019-05-30
Room	DMS 1160
Start time	10:00
Duration	00:45
INFO	
ID	1330
Event type	Lecture
Track	Hacking
Language used for presentation	English

FEEDBACK

Did you attend this event? Give Feedback

<<< >>>

Problem Statement

Allow users to

change the definition of an object **(DDL)**

while

the object is being used. **(Concurrent)**

But indexes are not user data objects?

Why things got complicated?

- Indexes are part of table description for planner

Why things got complicated?

- Indexes are part of table description for planner
- Indexes must contain references to all new row versions by the time CREATE INDEX transaction commit

Why things got complicated?

- Indexes are part of table description for planner
- Indexes must contain references to all new row versions by the time CREATE INDEX transaction commit
- Each transaction modifying data must update indexes

Virtual transaction

```
postgres=# begin;
postgres=*# select locktype,relation::regclass, virtualxid from pg_locks;
  locktype  | relation | virtualxid
------------+----------+------------
 relation   | pg_locks |
 virtualxid |          | 3/6
(2 rows)
```

Real transaction (Xid)

```
postgres=# begin;
postgres=*# SELECT txid_current();
postgres=*# select locktype, virtualxid, transactionid from pg_locks;
   locktype    | virtualxid | transactionid
---------------+------------+---------------
 relation      |            |
 virtualxid    | 3/8        |
 transactionid |            |           745
(3 rows)
```

Prepared transaction (2pc)

```
postgres=# begin;
postgres=*# insert into x values(1);
postgres=*# prepare transaction 'a';
postgres=# select relation::regclass, virtualtransaction from pg_locks
                where locktype = 'relation';
 relation | virtualtransaction
----------+--------------------
 pg_locks | 3/25
 x        | 3/20
(2 rows)
```

Lock levels

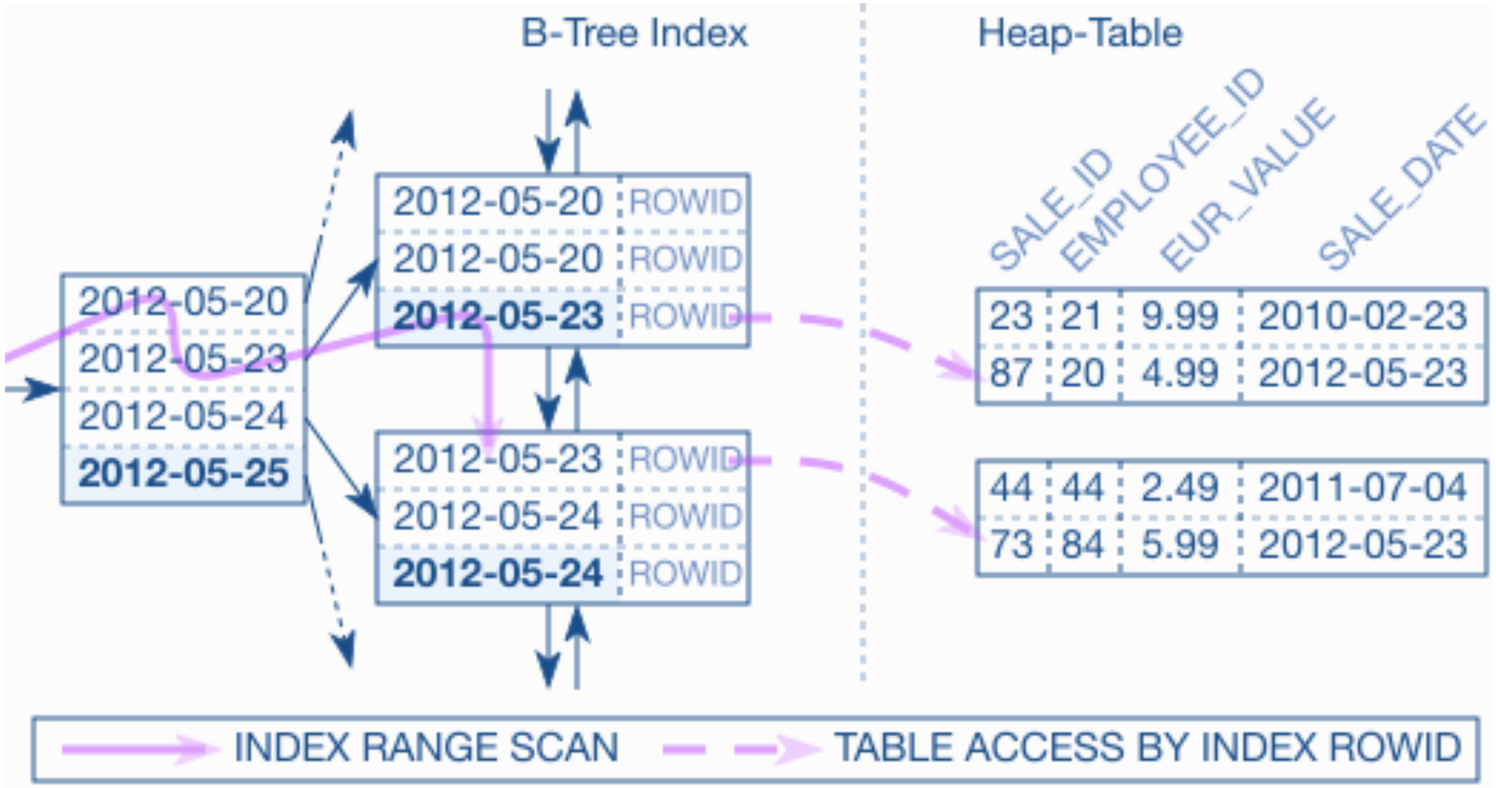
Lock level	Used by	Conflicts with
ACCESS SHARE	SELECT	ACCESS EXCLUSIVE
ROW SHARE	SELECT FOR SHARE \ SELECT FOR UPDATE	EXCLUSIVE
ROW EXCLUSIVE	UPDATE, DELETE, INSERT	SHARE
SHARE UPDATE EXCLUSIVE	VACUUM	SHARE UPDATE EXCLUSIVE
SHARE	CREATE INDEX	ROW EXCLUSIVE
SHARE ROW EXCLUSIVE		ROW EXCLUSIVE + Self
EXCLUSIVE		ROW SHARE
ACCESS EXCLUSIVE	ALTER TABLE, DROP TABLE и т.п.	ACCESS SHARE

https://www.postgresql.org/docs/14/explicit-locking.html
https://momjian.us/main/writings/pgsql/locking.pdf

Indexes

- Indexes should not affect results of the query, only performance
- So many access methods, shared infrastructure for all of them

Indexes must contain references to all new row versions by the time transaction commit



M. Winand: Use the Index, Luke https://use-the-index-luke.com/sql/clustering/index-organized-clustered-index

How to CREATE INDEX ?

- Get ShareLock on the table
- Create tuple about index in system catalog
- Invoke access method's *index_build()* routine
- Commit

Fully transactional operation

How to CREATE INDEX CONCURRENTLY?

1. Just build index somehow (even empty would work, but with less efficient resulting index)
2. Start inserting into it when new row versions are added
3. Validate index, ”catch up” missed row versions in (1)

Index vaildation

The actual extra work (besides locking)

Call *index_bulk_delete()* – but not for delete!

Collects ctids from index

Sorts ctids (row ids)

Call heapam_index_validate_scan()

SeqScan of heap table subtracting ctids already in index

How to CREATE INDEX CONCURRENTLY?

- Get ShareUpdateExclusiveLock on the table
- Create **invalid** index in the system catalog

Prevent incorrect HOT chains

- Invoke access method's *index_build()* routine
- Make index valid, bot not ready

Each transaction must create tuple in this index now

- Validate index: catch up old rows
- Mark index ready for index scans

Wait for old backends able to see too old tuples

How to CREATE INDEX CONCURRENTLY?

- Get ShareUpdateExclusiveLock on the table
- Create **invalid** index in the system catalog

WaitForLockers(heaplocktag, ShareLock)

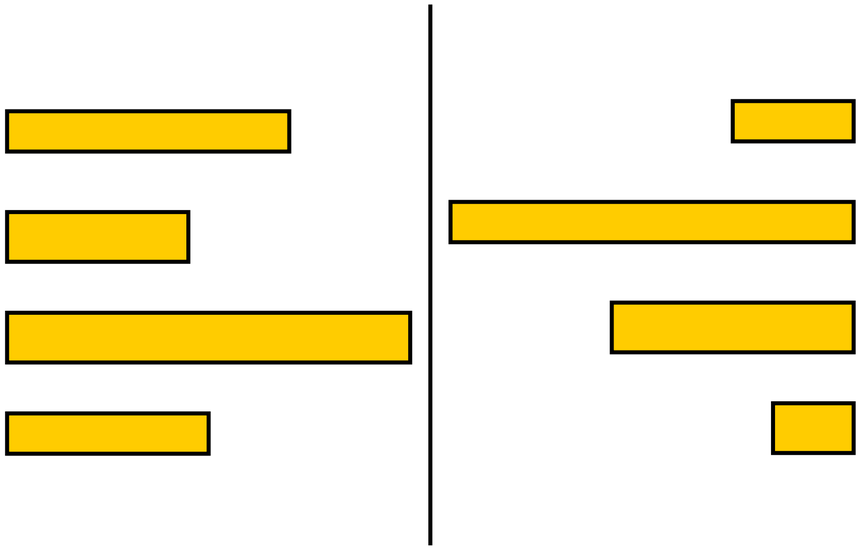
- Invoke access method's *index_build()* routine
- Make index valid, bot not ready

WaitForLockers(heaplocktag, ShareLock)

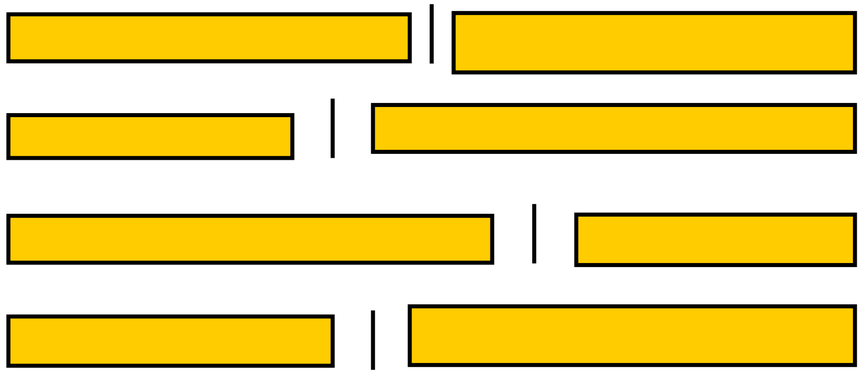
- Validate index: catch up old rows
- Mark index ready for index scans

WaitForOlderSnapshots(snapshot->xmin)

ShareLock “barrier”



table_open(relationId, ShareLock)



WaitForLockers(heaplocktag, ShareLock)

Lock holders

	Have vxid	Have xid	Lives in
vxid	v		ProcGlobal->allProcs[0:MaxConnections]
xid	v	v	ProcGlobal->allProcs[0:MaxConnections]
Prepared xid	v	v	PreparedXactProcs
Recovered prepared xid		v	PreparedXactProcs

```
lockholders = GetLockConflicts(Relation, ShareLock)
foreach holer  in lockholders
        VirtualXactLock(holder)
```

WaitForOlderSnapshots

```
/*
 * Wait for transactions that might have an older snapshot than the given xmin
 * limit, because it might not contain tuples deleted just before it has
 * been taken. Obtain a list of VXIDs of such transactions, and wait for them
 * individually. This is used when building an index concurrently.
 */
```

Scan through procArray and look for old backend.

REINDEX CONCURRENTLY

1. create new indexes in the catalog

2. build new indexes

3. let new indexes catch up with tuples inserted in the meantime

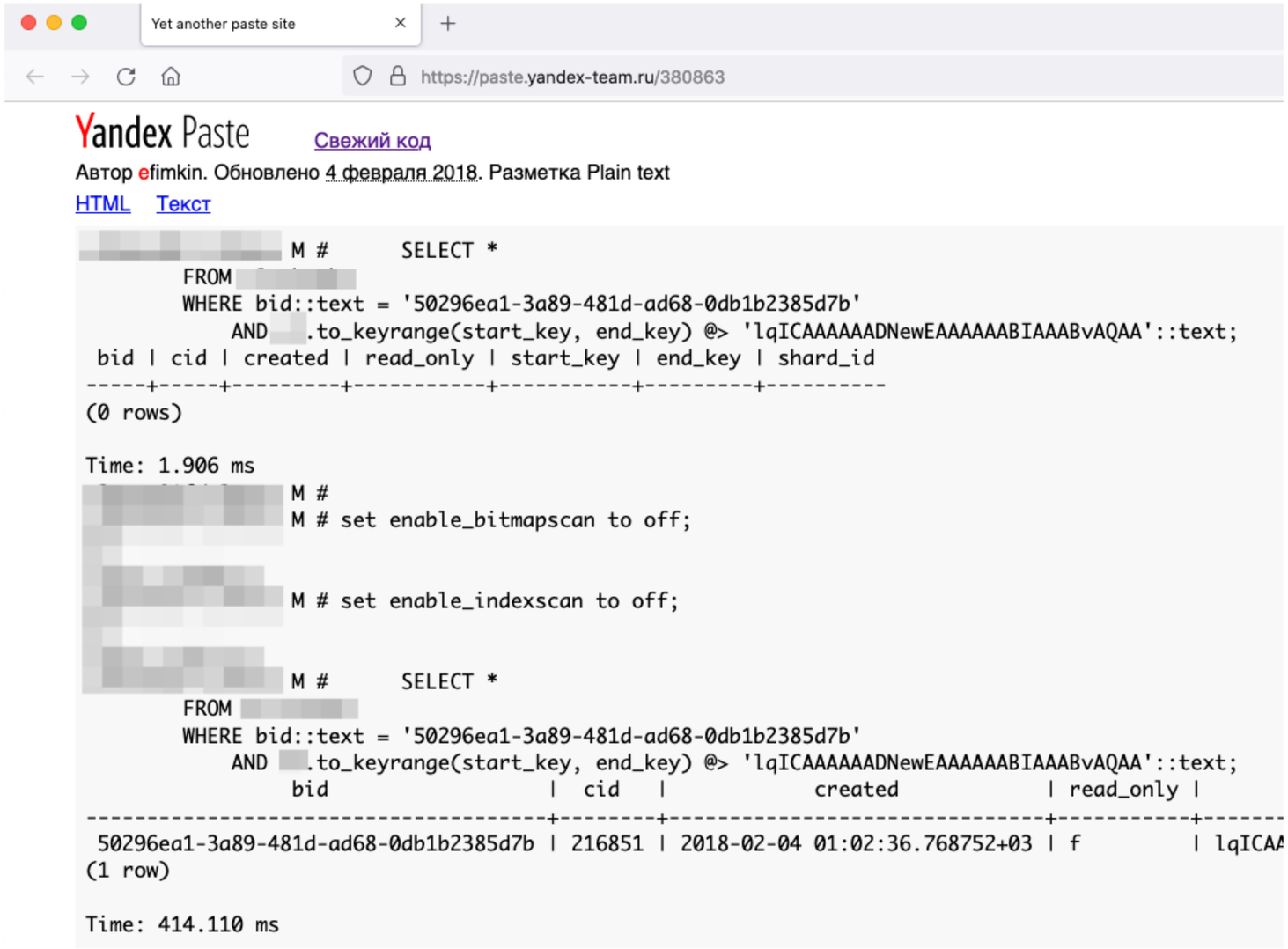
4. swap index names

5. mark old indexes as dead

6. drop old indexes

Hunting bugs

February 2018: first encounter



amcheck verification for GiST

Lists:pgsql-hackers

From: Andrey Borodin <x4mmm(at)yandex-team(dot)ru>
To: pgsql-hackers <pgsql-hackers(at)postgresql(dot)org>
Subject: amcheck verification for GiST
Date: 2018-09-23 10:15:09
Message-ID:59D0DA6B-1652-4D44-B0EF-A582D5824F83@yandex-team.ru
Views: Raw Message | Whole Thread | Download mbox | Resend email
Lists: pgsql-hackers

```
Hi, hackers!

Here's the patch with amcheck functionality for GiST.

It basically checks two invariants:
1. Every internal tuple need no adjustment by tuples of referenced page
2. Internal page reference or only leaf pages or only internal pages

We actually cannot check all balanced tree invariants due to concurrency reasons some concurrent splits will be visible as temporary balance violations.

Are there any other invariants that we can check?

I'd be happy to hear any thought about this.

Best regards, Andrey Borodin.
```

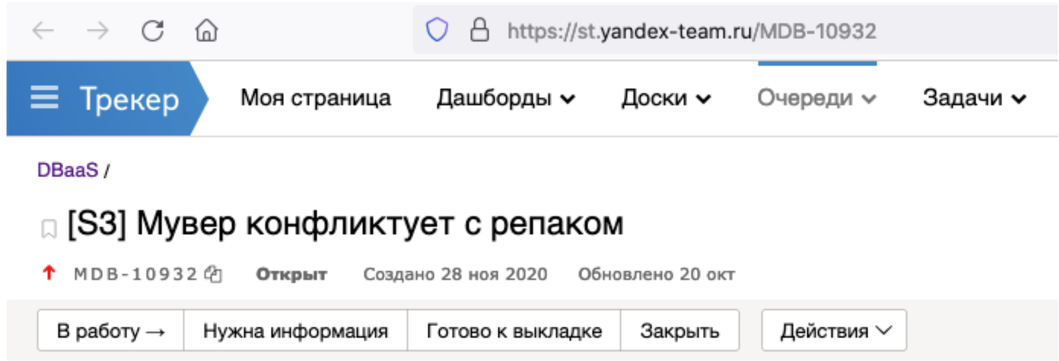
Attachment	Content-Type	Size
0001-GiST-verification-function-for-amcheck.patch	application/octet-stream	12.6 KB

November 2020: S3 metadata repack at risk

pg_repack --index is essentially REINDEX

We were doing it cron job every week

Saved up to 20% of disk space



Bug analysis vectors

- GiST is not a cause of a bug
- COPY from
- HOT chains
- Prepared statements
- System catalog caches

December 2020: reproduction

Андрей Бородин 16 дек 2020

Наблюадю воспроизведение
С таблицей

```
postgres=# \d t1
                Table "public.t1"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 i      | integer |           |          |
Indexes:
    "i1" btree (i)
```

Запускаю заполнение из Go

```
func doInserts(start int) {
        ctx := context.Background()
        conn, err := pgx.Connect(ctx, "host=127.0.0.1 port=5432 user=x4mmm database=postgres")
        pgConn := conn.PgConn()

        for i := start; ; i++ {
                is := fmt.Sprintf("%d", i)
                err = pgConn.Exec(ctx, fmt.Sprintf("begin;"+
                        "insert into t1 values(%d);"+
                        " PREPARE TRANSACTION 'x"+is+"'; ",
                        rand.Int31())).Close()
                exec := pgConn.Exec(ctx, " COMMIT PREPARED 'x"+is+"';")
        }
```

Запускаю бенч с CREATE INDEX CONCURRENTLY

```
drop index if exists i1; create index concurrently i1 on t1(i); select bt_index_check('i1',true);
```

Через несколько секунд появляется индекс в котором не хватает данных.

```
postgres=# select bt_index_check('i1',true);
NOTICE:  heap tuple (277,212) from table "t1" lacks matching index tuple within index "i1"
HINT:  Retrying verification using the function bt_index_parent_check() might provide a more specific error.
```

Prepared transactions

Own locks
Own xid

- Don’t have vxid

Lock holders

	Have vxid	Have xid	Lives in
vxid	v		ProcGlobal->allProcs[0:MaxConnections]
xid	v	v	ProcGlobal->allProcs[0:MaxConnections]
Prepared xid	v	v	PreparedXactProcs
Recovered prepared xid		v	PreparedXactProcs

```
lockholders = GetLockConflicts(Relation, ShareLock)
foreach holer  in lockholders
        VirtualXactLock(holder)
```

Lock holders

	Have vxid	Have xid	Lives in
vxid	v		ProcGlobal->allProcs[0:MaxConnections]
xid	v	v	ProcGlobal->allProcs[0:MaxConnections]
Prepared xid		v	PreparedXactProcs

```
lockholders = GetLockConflicts(Relation, ShareLock)
foreach holer  in lockholders
        VirtualXactLock(holder)
```

XID \ VXID barrier problem

GetLockConflicts() returns vxids

VirtualXactLock() accepts vxids

January 2020: Fix for 2PC

Home / Commitfest 2021-01 / CREATE INDEX CONCURRENTLY does not index prepared xact's data

CREATE INDEX CONCURRENTLY does not index prepared xact's data

Edit | Comment/Review ▾ | Change Status ▾

Title	CREATE INDEX CONCURRENTLY does not index prepared xact's data
Topic	Bug Fixes
Created	2020-12-20 18:13:07
Last modified	2021-01-30 16:04:49 (8 months, 3 weeks ago)
Latest email	2021-01-30 16:06:56 (8 months, 3 weeks ago)
Status	2021-01: Committed
Target version	
Authors	Andrey Borodin (x4m)
Reviewers	Noah Misch (nmisch)
Committer	Noah Misch (nmisch)
Links	

April 2021: Bug ressurection

From:	Andrey Borodin <x4mmm(at)yandex-team(dot)ru>
To:	Noah Misch <noah(at)leadboat(dot)com>
Cc:	Michael Paquier <michael(at)paquier(dot)xyz>, Tom Lane <tgl(at)sss(dot)pgh(dot)pa(dot)us>, pgsql-bugs(at)lists(dot)postgresql(dot)org
Subject:	Re: CREATE INDEX CONCURRENTLY does not index prepared xact's data
Date:	2021-05-01 12:42:25
Message-ID:	9C782C9D-D53E-4154-B910-46F062A9AE9B@yandex-team.ru
Views:	Raw Message | Whole Thread | Download mbox | Resend email
Thread:	2021-05-01 12:42:25 from Andrey Borodin <x4mmm(at)yandex-team(dot)ru>
Lists:	pgsql-bugs

```
> 30 янв. 2021 г., в 21:06, Andrey Borodin <x4mmm(at)yandex-team(dot)ru> написал(а):
>
>
>
>> 24 янв. 2021 г., в 07:27, Noah Misch <noah(at)leadboat(dot)com> написал(а):
>>
>> I changed that, updated comments, and fixed pgindent damage.  I plan to push
>> the attached version.
>
> I see that patch was pushed. I'll flip status of CF entry. Many thanks!

FWIW I have 2 new reported cases on 12.6. I've double-checked that at the moment of corruption installation run version with the patch.
To the date I could not reproduce the problem myself, but I'll continue working on this.

Thanks!

Best regards, Andrey Borodin.
```

Analysis

- VXID -> Backend XID -> 2PC XID

May 2021: Reproduction

```
begin;
insert into t1 values(0);

-- session C: reindex table concurrently t1;

prepare transaction 'a';
begin;
insert into t1 values(0);
-- session B: commit prepared 'a';
prepare transaction 'b';
begin;
insert into t1 values(0);
-- session B: commit prepared 'b';
prepare transaction 'c';

begin;
insert into t1 values(0);
-- session B: commit prepared 'c';
prepare transaction 'd';
commit prepared 'd';

-- session C: postgres=# select bt_index_check('i1',true);
ERROR:  heap tuple (0,2) from table "t1" lacks matching index tuple within index "i1"
```

June 2021: TAP tests with pgbench

- REINDEX CONCURRENTLY deadlocks with itself

June 2021: TAP tests with pgbench

```
# Run pgbench.
$node->pgbench(
        '--no-vacuum --client=5 --time=10',
        0,
        [qr{actually processed}],
        [qr{^$}],
        'concurrent INSERTs w/ 2PC',
        {
                '002_pgbench_concurrent_2pc' => q(
                        BEGIN;
                        INSERT INTO tbl VALUES(0);
                        PREPARE TRANSACTION 'c:client_id';
                        COMMIT PREPARED 'c:client_id';
                  ),
                '002_pgbench_concurrent_2pc_savepoint' => q(
                        BEGIN;
                        SAVEPOINT s1;
                        INSERT INTO tbl VALUES(0);
                        PREPARE TRANSACTION 'c:client_id';
                        COMMIT PREPARED 'c:client_id';
                   )
        });
```

Analysis, anger, bargaining, depression, acceptance

From:	Andrey Borodin <x4mmm(at)yandex-team(dot)ru>
To:	Noah Misch <noah(at)leadboat(dot)com>
Cc:	Michael Paquier <michael(at)paquier(dot)xyz>, Tom Lane <tgl(at)sss(dot)pgh(dot)pa(dot)us>, pgsql-bugs(at)lists(dot)postgresql(dot)org
Subject:	Re: CREATE INDEX CONCURRENTLY does not index prepared xact's data
Date:	2021-07-27 12:50:55
Message-ID:	845C714B-DBCF-49E9-9F95-238DB042F926@yandex-team.ru
Views:	Raw Message | Whole Thread | Download mbox | Resend email
Lists:	pgsql-bugs

```
> 24 июля 2021 г., в 03:30, Noah Misch <noah(at)leadboat(dot)com> написал(а):
>
> It could be okay, but I think it's better to add the test under amcheck.  You
> could still use pgbench in the test.
Currently it's still WIP.
I've added two tests: deterministic with psql and probabilistic with pgbench.
And I really do not like pgbench test:
1. It does not seem stable enough, it can turn buildfarm red as a good watermelon.
2. Names for 2PCs are choosen at random and have probability of collision.
3. It still breaks the fix and I have no idea why.

Can you please take a look on added TAP test? Probably I'm doing wrong a lot of things, it's the very first program on Perl written by me...
background_pgbench is a local invention. sub pgbench is a copy from nearby test. Should I refactor it somewhere?

Thanks!

Best regards, Andrey Borodin.
```

Attachment	Content-Type	Size
v5-0001-Introduce-TAP-test-for-2PC-with-CIC-behavior.patch	application/octet-stream	7.0 KB
v5-0002-Fix-CREATE-INDEX-CONCURRENTLY-in-precence-of-vxid.patch	application/octet-stream	8.2 KB

Relation cache

```
static Relation
RelationBuildDesc(Oid targetRelId, bool insertIt)
{
        int                     in_progress_offset;
        Relation        relation;
        Oid                     relid;
        HeapTuple       pg_class_tuple;
        Form_pg_class relp;

        /*
         * This function and its subroutines can allocate a good deal of transient
         * data in CurrentMemoryContext.  Traditionally we've just leaked that
         * data, reasoning that the caller's context is at worst of transaction
         * scope, and relcache loads shouldn't happen so often that it's essential
         * to recover transient data before end of statement/transaction.  However
         * that's definitely not true when debug_discard_caches is active, and
         * perhaps it's not true in other cases.
```

Using a table description in transaction lifecycle

Take a lock on table
Check shared invalidations (sinval)
Check if relation in relcache

› Put relation description to cache

...
On commit send invalidation (sinval) if DDL happened
Release lock

Using a table description in transaction lifecycle

Take a lock on table
Check shared invalidations (sinval)
Check if relation in relcache

› Put relation description to cache

Another sinval arrived

...
On commit send invalidation (sinval) if DDL happened
Release lock

August 2021: Fix by Noah Misch

```c
/*
 * in_progress_list is a stack of ongoing RelationBuildDesc() calls.  CREATE
 * INDEX CONCURRENTLY makes catalog changes under ShareUpdateExclusiveLock.
 * It critically relies on each backend absorbing those changes no later than
 * next transaction start.  Hence, RelationBuildDesc() loops until it finishes
 * without accepting a relevant invalidation.  (Most invalidation consumers
 * don't do this.)
 */
typedef struct inprogressent
{
        Oid                     reloid;                 /* OID of relation being built */
        bool            invalidated;    /* whether an invalidation arrived for it */
} InProgressEnt;

static InProgressEnt *in_progress_list;
static int      in_progress_list_len;
static int      in_progress_list_maxlen;
```

September 2021: XID lock

```
s->fullTransactionId = GetNewTransactionId(isSubXact);
....
XactLockTableInsert(XidFromFullTransactionId(s->fullTransactionId));
```

If we have vxid – we need to lock vxid

And one more problem

- In PrepareTransaction()

ProcArrayClearTransaction(MyProc);

PostPrepare_Locks(xid);

And one more fix

- In PrepareTransaction()

PostPrepare_Locks(xid);

ProcArrayClearTransaction(MyProc);

ShareLock barrier

- Get conflicting *VXIDs*, if no *VXID* in PGPROC then take **XID**
- Wait for all *VXIDs*, wait for all **XIDs**
- For **XID** without known *VXID* search among 2PC

Wait for all newly found XIDs

October 2021: Fix pushed

Noah Misch pgsql-committers 04:40

pgsql: Fix CREATE INDEX CONCURRENTLY for the newest prepared transactio

Кому: pgsql-committers

```
Fix CREATE INDEX CONCURRENTLY for the newest prepared transactions.

The purpose of commit 8a54e12a38d1545d249f1402f66c8cde2837d97c was to
fix this, and it sufficed when the PREPARE TRANSACTION completed before
the CIC looked for lock conflicts.  Otherwise, things still broke.  As
before, in a cluster having used CIC while having enabled prepared
transactions, queries that use the resulting index can silently fail to
find rows.  It may be necessary to reindex to recover from past
occurrences; REINDEX CONCURRENTLY suffices.  Fix this for future index
builds by making CIC wait for arbitrarily-recent prepared transactions
and for ordinary transactions that may yet PREPARE TRANSACTION.  As part
of that, have PREPARE TRANSACTION transfer locks to its dummy PGPROC
before it calls ProcArrayClearTransaction().  Back-patch to 9.6 (all
supported versions).

Andrey Borodin, reviewed (in earlier versions) by Andres Freund.

Discussion: https://postgr.es/m/01824242-AA92-4FE9-9BA7-AEBAFFEA3D0C@yandex-team.ru

Branch
------
master
```

But still there are some problems

From:	Noah Misch <noah(at)leadboat(dot)com>
To:	Andrey Borodin <x4mmm(at)yandex-team(dot)ru>
Cc:	PostgreSQL mailing lists <pgsql-bugs(at)lists(dot)postgresql(dot)org>, Michael Paquier <michael(at)paq
Subject:	Re: CREATE INDEX CONCURRENTLY does not index prepared xact's data
Date:	2021-10-24 05:00:28
Message-ID:	20211024050028.GA3310519@rfd.leadboat.com
Views:	Raw Message | Whole Thread | Download mbox | Resend email
Lists:	pgsql-bugs

```
On Mon, Oct 18, 2021 at 08:02:12PM -0700, Noah Misch wrote:
> On Mon, Oct 18, 2021 at 06:23:05PM +0500, Andrey Borodin wrote:
> > > 17 окт. 2021 г., в 20:12, Noah Misch <noah(at)leadboat(dot)com> написал(а):
> > > I think the attached version is ready for commit.  Notable differences
> > > vs. v14:

Pushed.  Buildfarm member conchuela (DragonFly BSD 6.0) has gotten multiple
"IPC::Run: timeout on timer" in the new tests.  No other animal has.
https://buildfarm.postgresql.org/cgi-bin/show_log.pl?nm=conchuela&dt=2021-10-24%2003%3A05%3A09
is an example run.  The pgbench queries finished quickly, but the
$pgbench_h->finish() apparently timed out after 180s.  I guess this would be
consistent with pgbench blocking in write(), waiting for something to empty a
pipe buffer so it can write more.  I thought finish() will drain any incoming
I/O, though.  This phenomenon has been appearing regularly via
src/test/recovery/t/017_shm.pl[1], so this thread doesn't have a duty to
resolve it.  A stack trace of the stuck pgbench should be informative, though.
```

The most complex bug I ever worked on

- Reproduced in production once in 3 month with 1 MRPS avg workload
- 3 Yandex services reported suspicious behavior from Feb 2018
- 9 month of active investigation knowing where to search and what to fix
- Parts of the fix committed 4 times, 6 reviewers
- Tests triggered kernel bugs in 2 OSes

February 2022: To be continued on Standby

PG Bug reporting form 9 February 2022, 02:01

BUG #17401: REINDEX TABLE CONCURRENTLY creates a race condition on a streaming replica

To: PostgreSQL mailing lists, Cc: Ben Chobot Details

The following bug has been logged on the website:

Bug reference: 17401
Logged by: Ben Chobot
Email address: bench@silentmedia.com
PostgreSQL version: 12.9
Operating system: Linux (Ubuntu)
Description:

This bug is is almost identical to BUG #17389, which I filed blaming pg_repack; however, further testing shows the same symptoms using vanilla REINDEX TABLE CONCURRENTLY.

Questions? ☺

Andrey Borodin

 x4mmm @yandex-team.ru

 x4mmm