PGCon 2023

Global Unique Index

A Different Approach

Cary Huang
Highgo Software (Canada)

In this talk

PGCon 2023

HIGH GO

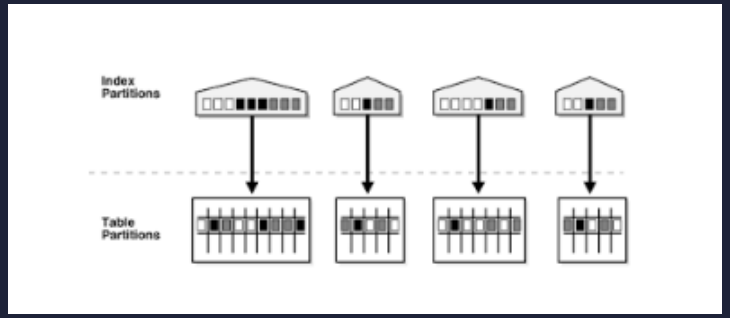
- Few words about me
- Global index – Background information, benefit and drawbacks
- Global unique index – Introduction to our approach
- Global unique index – Syntax
- Global unique index – How it works
- Global unique index – Benchmark numbers
- Global unique index – The Lock Problem
- Summary
- Related links

Few Words About Me: Cary

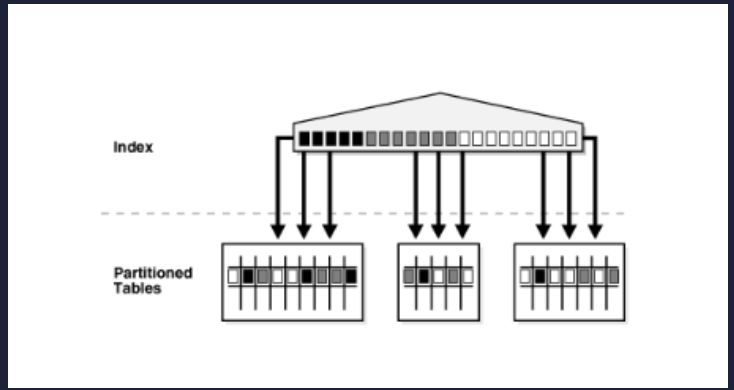
- Bachelor of Electrical Engineering graduate from University of British Columbia (UBC) In Vancouver in 2012
- Worked as a software developer & team lead in a smart metering innovation company after graduation
- Joined Highgo Software in 2019 to start my PostgreSQL journey
- Post-graduate instructor at Peking University for open-source projects in 2021 (based on PostgreSQL of course)
- Worked on several aspects of PG including sharding enhancement, distributed database, HA, shared storage, security…etc

Global Index – Background Information

- First proposed in 2019 with title “Proposal: Global Index”
- Only applicable to partitioned tables/indexes
- One index relation at global scale that indexes all child table partitions. (1 to many)
- Kind of combine all partitions into one



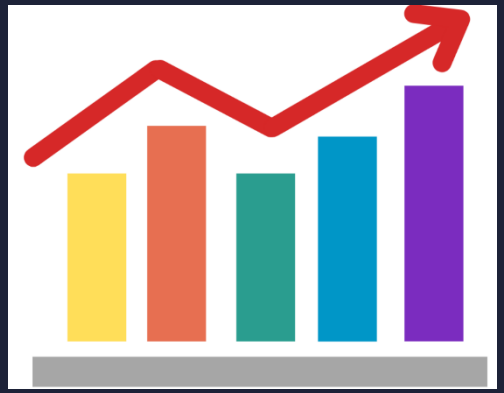
Regular Index



Global Index

Images source:
https://www.filibeto.org/sun/lib/nonsun/oracle/11.1.0.6.0/B28359_01/server.111/b32024/partition.htm

Global Index – Benefit



- Cross-partition uniqueness guarantee
- Performance increase
- Partitioned keys no longer required to include when creating a unique index

Global Index – Drawbacks

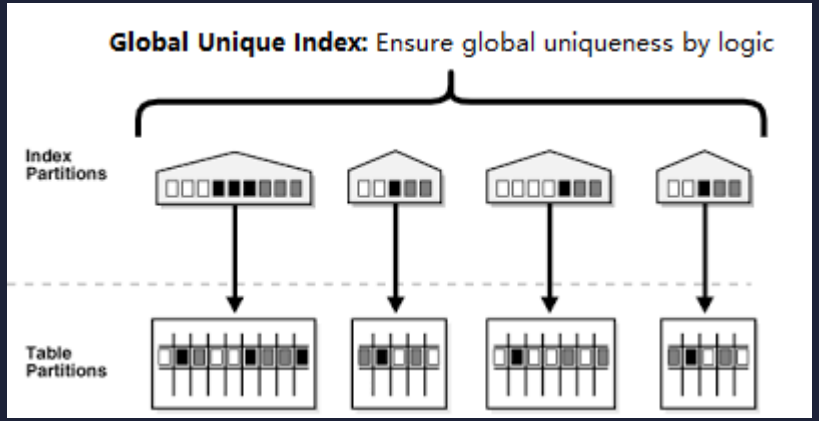
- Architectural changes needed
- A large global index might reintroduce problems that large relations already have
  - Slower vacuum and maintenance
  - Slower indexing
  - … etc
- That kind of contradicts the purpose of having a partitioned table.

Global Unique Index – our Approach

PGCon 2023

HIGH GO

- Keep in mind the difference in terminology: “Global Index” vs “Global Unique Index”
- To achieve the same benefit without invasive changes to partition table architecture.
- We add new logics to achieve cross-partition uniqueness
- Architecturally the same as regular unique index except one can guarantee cross-partition uniqueness check, the other cannot.



Global Unique Index – Syntax

- A new clause “GLOBAL” to be used in conjunction with “CREATE UNIQUE INDEX”
- No longer need to include all partition keys when creating a unique index

```
> CREATE TABLE gidx_part (a int, b int, c text) PARTITION BY RANGE (a);
> CREATE TABLE gidx_part1 partition of gidx_part FOR VALUES FROM (0) TO (10);
> CREATE TABLE gidx_part2 partition of gidx_part FOR VALUES FROM (10) TO (20);
> INSERT INTO gidx_part values(5, 5, 'test’);
> INSERT INTO gidx_part values(15, 5, 'test');
> CREATE UNIQUE INDEX global_unique_idx ON gidx_part USING BTREE(b) GLOBAL;
ERROR:  could not create unique index "global_unique_idx"
DETAIL:  Key (bid)=(5) is duplicated.
```

No partition key here

Global Unique Index – Without GLOBAL

- Baseline PG requires inclusion of all partition keys when creating unique index (multi-column index)
- Lifting this constraint results in increased query performance (more later)

```
> CREATE TABLE gidx_part (a int, b int, c text) PARTITION BY RANGE (a);
> CREATE TABLE gidx_part1 partition of gidx_part FOR VALUES FROM (0) TO (10);
> CREATE TABLE gidx_part2 partition of gidx_part FOR VALUES FROM (10) TO (20);
> INSERT INTO gidx_part values(5, 5, 'test');
> INSERT INTO gidx_part values(15, 5, 'test');
> CREATE UNIQUE INDEX global_unique_idx ON gidx_part USING BTREE(b);
ERROR:  unique constraint on partitioned table must include all partitioning columns
DETAIL:  UNIQUE constraint on table "gidx_part" lacks column "a" which is part of the partition key.
```

Needs (a, b) here without global.

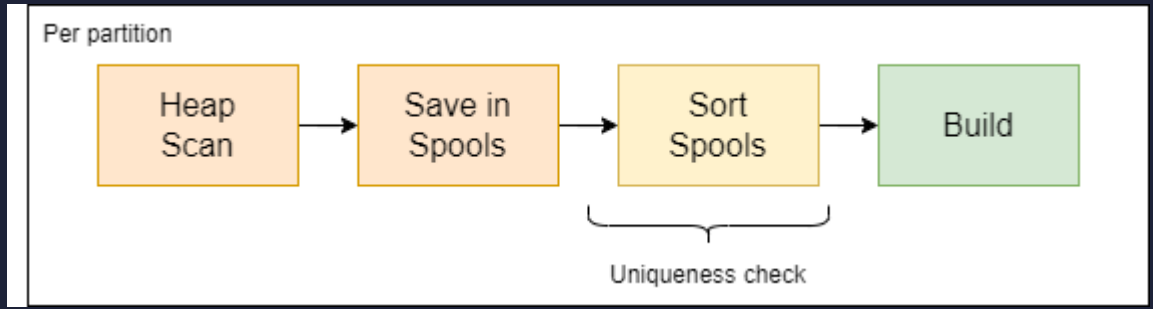
Global Unique Index – New Relkind

- Unique index created with “GLOBAL” clause will have a new relkind = “g”
- This is the main identifier for the global uniqueness implementation

```
postgres=# update pgbench_accounts set bid=aid;
UPDATE 100000
postgres=# CREATE UNIQUE INDEX global_unique_idx ON pgbench_accounts USING BTREE(bid) GLOBAL;
CREATE INDEX
postgres=# select oid, relname, relfilenode, relpages, reltuples, relhasindex, relkind from pg_cl
ass where relname like 'pgbench_account%' or relname like 'global_unique_idx' order by oid;
  oid  |          relname           | relfilenode | relpages | reltuples | relhasindex | relkind
-------+----------------------------+-------------+----------+-----------+-------------+---------
 16390 | pgbench_accounts           |           0 |       -1 |    100000 | t           | p
 16396 | pgbench_accounts_1         |       16405 |     1093 |     33334 | t           | r
 16399 | pgbench_accounts_2         |       16406 |     1093 |     33334 | t           | r
 16402 | pgbench_accounts_3         |       16407 |     1093 |     33332 | t           | r
 16415 | pgbench_accounts_pkey      |           0 |        0 |         0 | f           | I
 16417 | pgbench_accounts_1_pkey    |       16417 |       94 |     33334 | f           | i
 16419 | pgbench_accounts_2_pkey    |       16419 |       94 |     33334 | f           | i
 16421 | pgbench_accounts_3_pkey    |       16421 |       94 |     33332 | f           | i
 16423 | global_unique_idx          |           0 |        0 |         0 | f           | I
 16424 | pgbench_accounts_1_bid_idx |       16424 |       94 |     33334 | f           | g
 16425 | pgbench_accounts_2_bid_idx |       16425 |       94 |     33334 | f           | g
 16426 | pgbench_accounts_3_bid_idx |       16426 |       94 |     33332 | f           | g
```

How Regular Unique Index Create Works

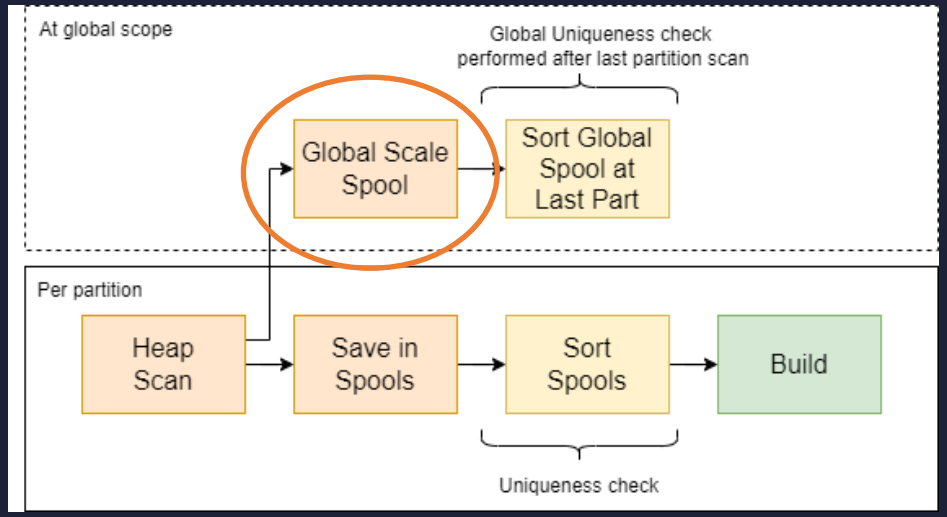
- Btree Index creation requires heap scan, scanning all tuples into “spool” structure
- Based on index key, perform sorting within spool and then a btree is constructed from sorted spool.
- Uniqueness check happens during sorting.
- For partitioned table index creation, this process is repeated per partition
- Cannot guarantee cross-partition uniqueness



How Global Unique Index Create Works

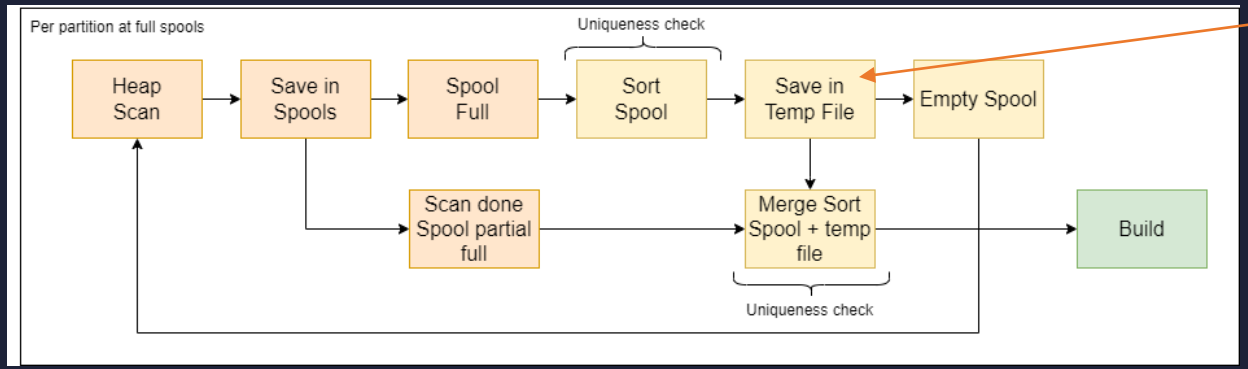
- If we heap scan all partitions into a single & global-scale “spool” structure and then do a “final sorting” after the last partition scan, can we achieve global uniqueness?

- Indeed yes, but it also creates new problems:
  - How much data can a spool structure hold?
  - What if parallelism is used to create the global unique index



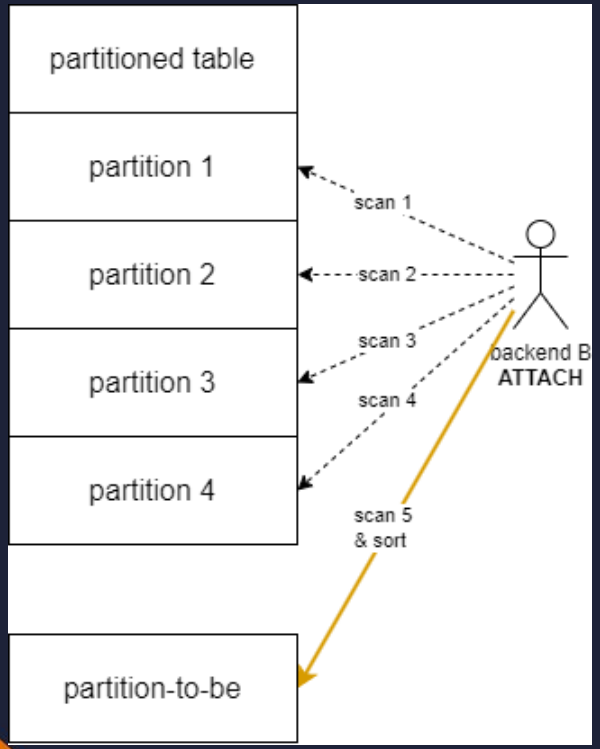
Problems with this Approach

- Size of spool?
  - Specified by “maintenance_work_mem” (default 64MB)
  - When full, PG switches to “tape-based sorting”, meaning, it will sort on current spool, save the results in a temporary file and empties the spool.
  - Finally, it performs a “merge sort” of all temp files while building the final btree.
- Create global unique index in parallel?
  - Also achieved by tape-based sorting, worker writes sort results on file, leader does the merge sort



Also known as “logical tapes”

Global Unique Index – ATTACH



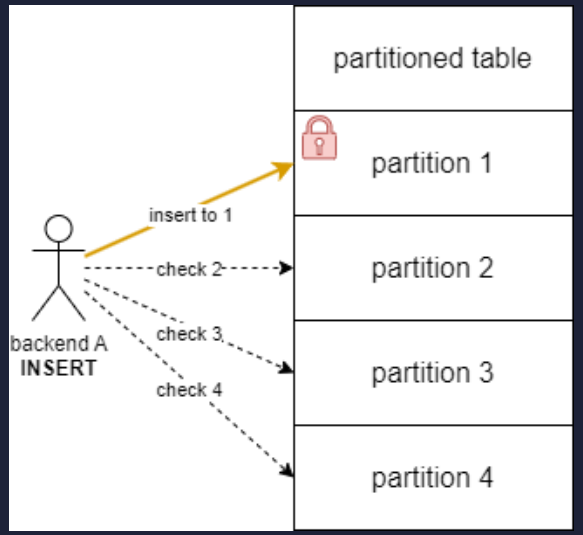
- Requires more work than regular unique index attach
- If the table has no data, we can attach right away
- If it has data, we need to utilize the “spool” and “sorting” mechanism again to ensure global uniqueness.

Global Unique Index – DETACH

- Detach a partition with global unique index is relatively easy
- We simply go through the same detach process that current PG already has
- Except that at the end of the detach, we will transform all global unique index into regular unique index
- This is done simply by changing the relkind from ‘g’ to ‘i’.

Global Unique Index – INSERT & UPDATE

- For every tuple inserted or updated, PG attempts to fetch the same tuple with the same unique index key from the current partition to determine if it violates uniqueness.
- With a global unique index, this fetch is extended to all "other partitions" as well
- Adds extra performance cost (has room for future optimization)



Global Unique Index – Timings

Global Unique Index

- 156285ms to insert
- 6592ms to delete
- 3957ms to create
- 3650ms to attach
- 17ms to detach

Regular Unique Index

- 26007ms to insert
- 6738ms to delete
- 2933ms to create
- 628ms to attach
- 17ms to detach

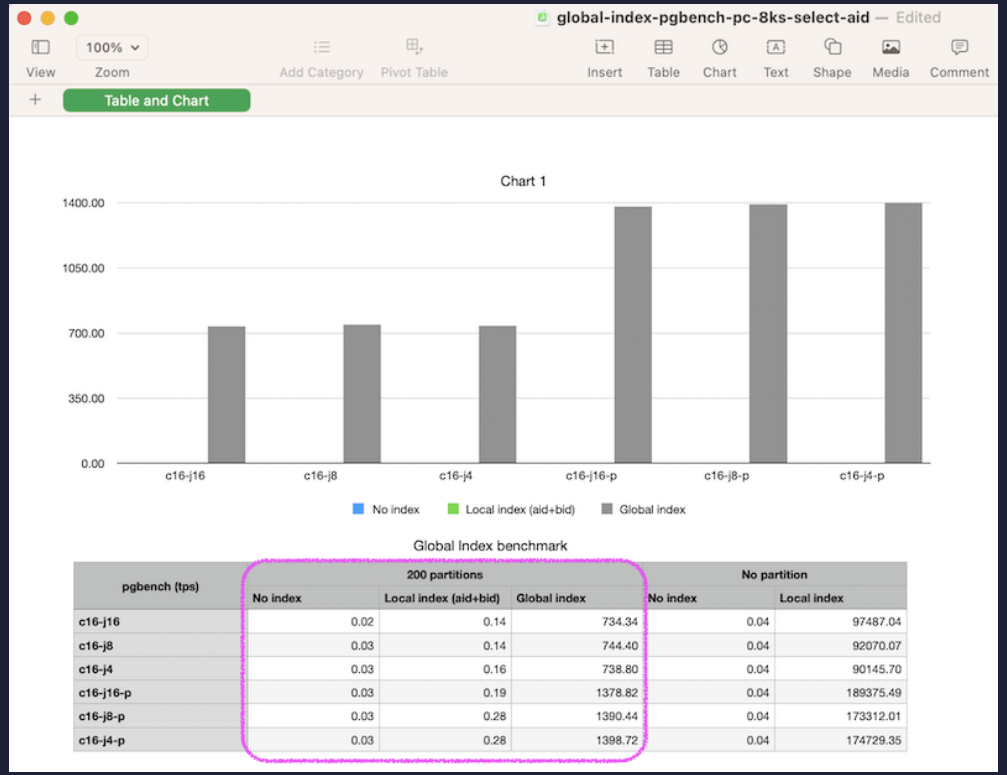
```
> create table test(a int, b int, c text) partition by range (a);
> create table test1 partition of test for values from (MINVALUE) to (1000000);
> create table test2 partition of test for values from (1000000) to (2000000);
> create table test3 partition of test for values from (2000000) to (3000000);
> create table test4 partition of test for values from (3000000) to (4000000);
> create table test5 partition of test for values from (4000000) to (5000000);
> create table test6 partition of test for values from (5000000) to (6000000);

> create unique index myindex on test(b) global;
> insert into test values(generate_series(0,5999999), generate_series(0,5999999), 'test');
> delete from test;
> drop index myindex;
> insert into test values(generate_series(0,5999999), generate_series(0,5999999), 'test');
> create unique index myindex on test(b) global;
> create table test7 (a int, b int, c text);
> insert into test7 values(generate_series(6000000, 6999999),
  generate_series(6000000, 6999999), 'test');
> alter table test attach partition test7 for values from (6000000) TO (7000000);
> alter table test detach partition test7;
```

Performance loss for insert & attach are proportional to number of partitions

Benchmark Number – SELECT Only Using Unique Column as Lookup Key

- Test condition with pgbench:
  - 800,000,000 records (110GB)
  - 200 partitions
  - SELECT only
- Results
  - No index: 0.02 tps
  - Unique index (aid + bid): 0.14 tps
  - Global unique index (bid):

    734 ~ 1398 tps



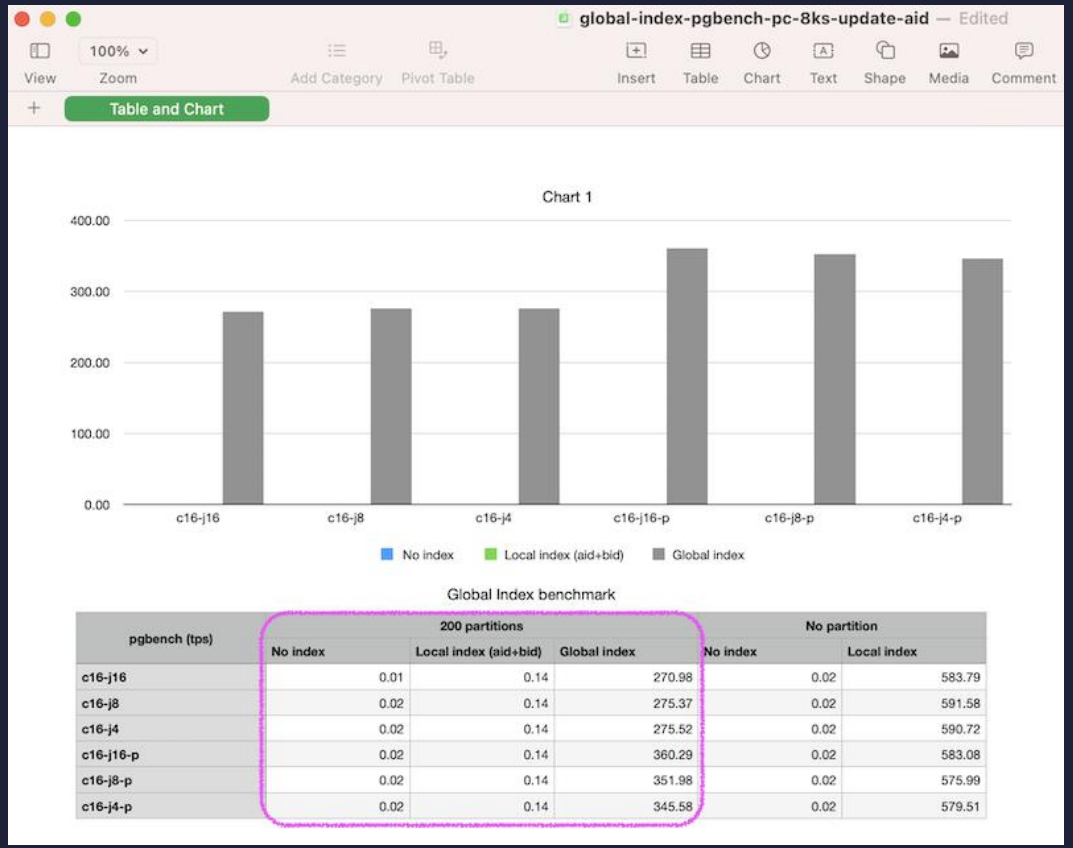
Global Index benchmark

pgbench (tps)	200 partitions			No partition	
	No index	Local index (aid+bid)	Global index	No index	Local index
c16-j16	0.02	0.14	734.34	0.04	97487.04
c16-j8	0.03	0.14	744.40	0.04	92070.07
c16-j4	0.03	0.16	738.80	0.04	90145.70
c16-j16-p	0.03	0.19	1378.82	0.04	189375.49
c16-j8-p	0.03	0.28	1390.44	0.04	173312.01
c16-j4-p	0.03	0.28	1398.72	0.04	174729.35

Benchmark Number – SELECT + UPDATE

- Same test condition with pgbench
- Results
  - No index: 0.02 tps
  - Unique index (aid + bid): 0.14 tps
  - Global unique index (bid):

    270 ~ 360 tps



Global Index benchmark

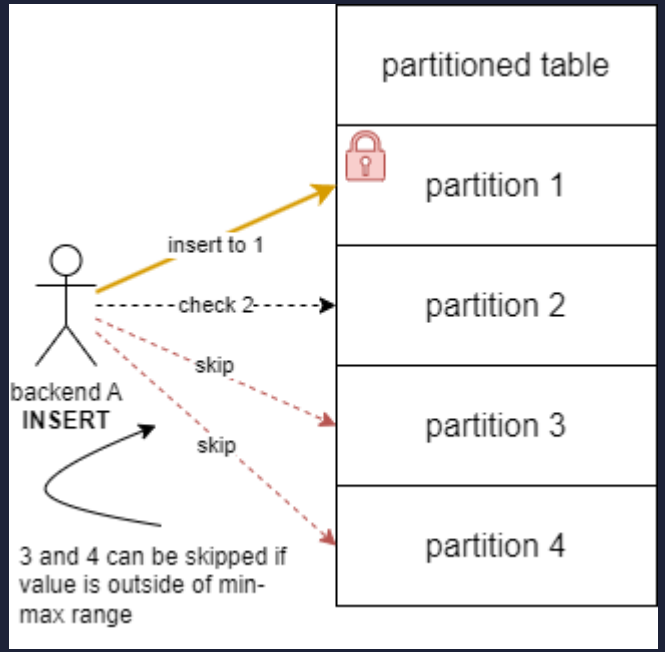
pgbench (tps)	200 partitions			No partition	
	No index	Local index (aid+bid)	Global index	No index	Local index
c16-j16	0.01	0.14	270.98	0.02	583.79
c16-j8	0.02	0.14	275.37	0.02	591.58
c16-j4	0.02	0.14	275.52	0.02	590.72
c16-j16-p	0.02	0.14	360.29	0.02	583.08
c16-j8-p	0.02	0.14	351.98	0.02	575.99
c16-j4-p	0.02	0.14	345.58	0.02	579.51

Performance Impact Summary

- INSERT and ATTACH have the most performance degradation and is directly proportional to number of partitions
- CREATE is about 35% slower due to maintaining a separate “spool” structure
- DELETE and DETACH are roughly the same with or without global unique index
- SELECT on global unique index is multiple-magnitude faster than traditional unique index due to the removal of partition keys during CREATE.

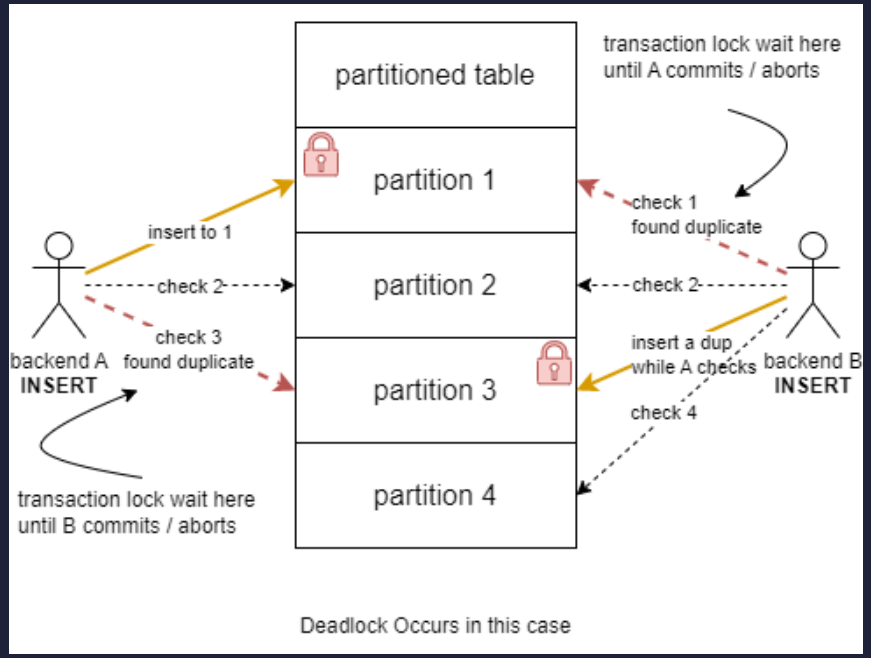
Possible Ways to Improve Performance on INSERT

- INSERT is the slowest because it must visit all other partitions for uniqueness check
- We could maintain a max and min value(s) mapping per partition that are updated when a partition is modified.
- Based on this min and max values, the system could avoid visiting a partition if the unique key value to be inserted is outside of the min – max range.
- An idea from community, have not tried it yet :p



Global Unique Index – Lock Problems For INSERT

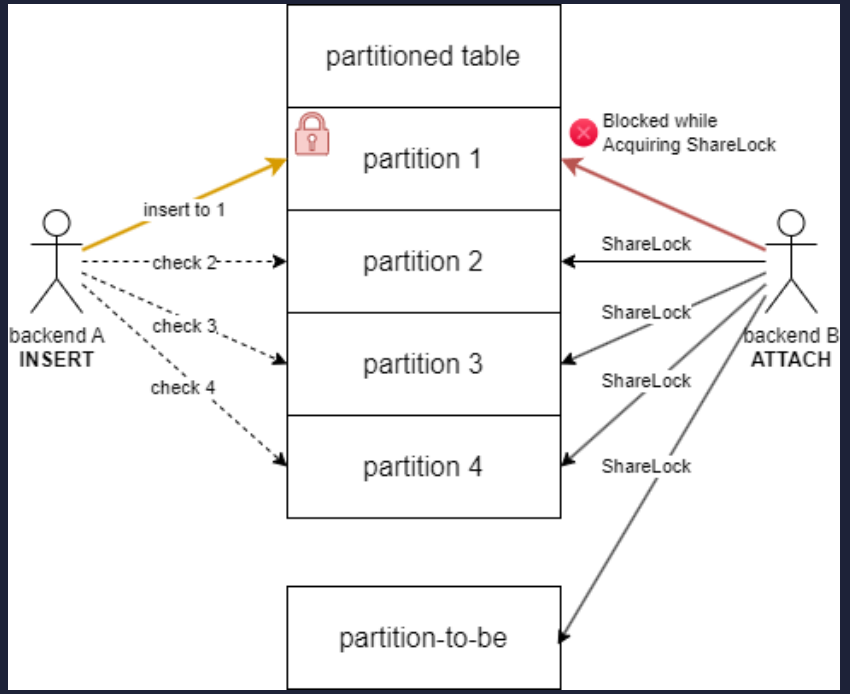
- To INSERT, backend A acquires AccessShareLock on all other partitions to check uniqueness. This allows backend B to insert a duplicate right after backend A finishes its checking.
- A "transaction level lock" will be triggered when this happens
- Possible deadlock could be triggered when both backend A and B detects each other's duplicate at the same time.



May be okay, because this happens when there is a global conflict detected. It is going to error out anyway

Global Unique Index – Lock Problems For ATTACH

- ATTACH tries to acquire ShareLock on all existing partitions and partition-to-be for uniqueness checks.
- If there are concurrent inserts prior to attach, some partitions may be exclusively locked, causing attach to wait.
- If ATTACH happens first, the INSERTs will have to wait instead
- This locking limits the flexibility of ATTACH



Global Unique Index – Possible Solution to ATTACH Lock Problems

- We cannot lower ATTACH's lock level to AccessShareLock
- A duplicate can always be inserted after ATTACH finishes checking a partition without proper locking
- There is no "transaction level lock" involved during attach
- We could perhaps add "ATTACH CONCURRENTLY" feature similar to "CREATE INDEX CONCURRENTLY for partitioned tables" that uses the same principle but for ATTACH.
- More details can be found here:

https://www.postgresql.org/message-id/20201031063117.GF3080%40telsasoft.com

Global Unique Index - Summary

The good:

- Cross-partition uniqueness guarantee
- Significant increase in SELECT performance on unique key columns
- Inclusion of partition keys no longer enforced
- No architecture change to partitioned table, optimizer and planner

The bad:

- INSERT and ATTACH are slow with more number of partitions (have room for optimization)
- ATTACH needs to lock all partitions from concurrent insertions to ensure uniqueness
- Possible deadlock with concurrent INSERTs

Related Links

PGCon 2023

HIGH GO

- Original discussion on global index
  - https://www.postgresql.org/message-id/CALtqXTcurqy1PKXzP9XO%3DofLLA5wBSo77BnUnYVEZpmcA3V0aq%40mail.gmail.com
- Discussion on global unique index
  - https://www.postgresql.org/message-id/184879c5306.12490ea581628934.7312528450011769010@highgo.ca
- Related blogs
  - https://www.highgo.ca/2022/10/14/global-index-a-different-approach/
  - https://www.highgo.ca/2022/10/28/cross-partition-uniqueness-guarantee-with-global-unique-index/
  - https://www.highgo.ca/2022/11/25/global-index-benchmark-with-pgbench/

Thank You

Any Question?

You can email Cary at:
cary.huang@highgo.ca

融知与行 瀚且高远

Knowledge and action constitute to immense accomplishment

THANKS