POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20

A couple of new features towards the sharding

LEPIKHOV Andrey
Core Developer
Postgres Professional

https://2020.postgresconf.cn

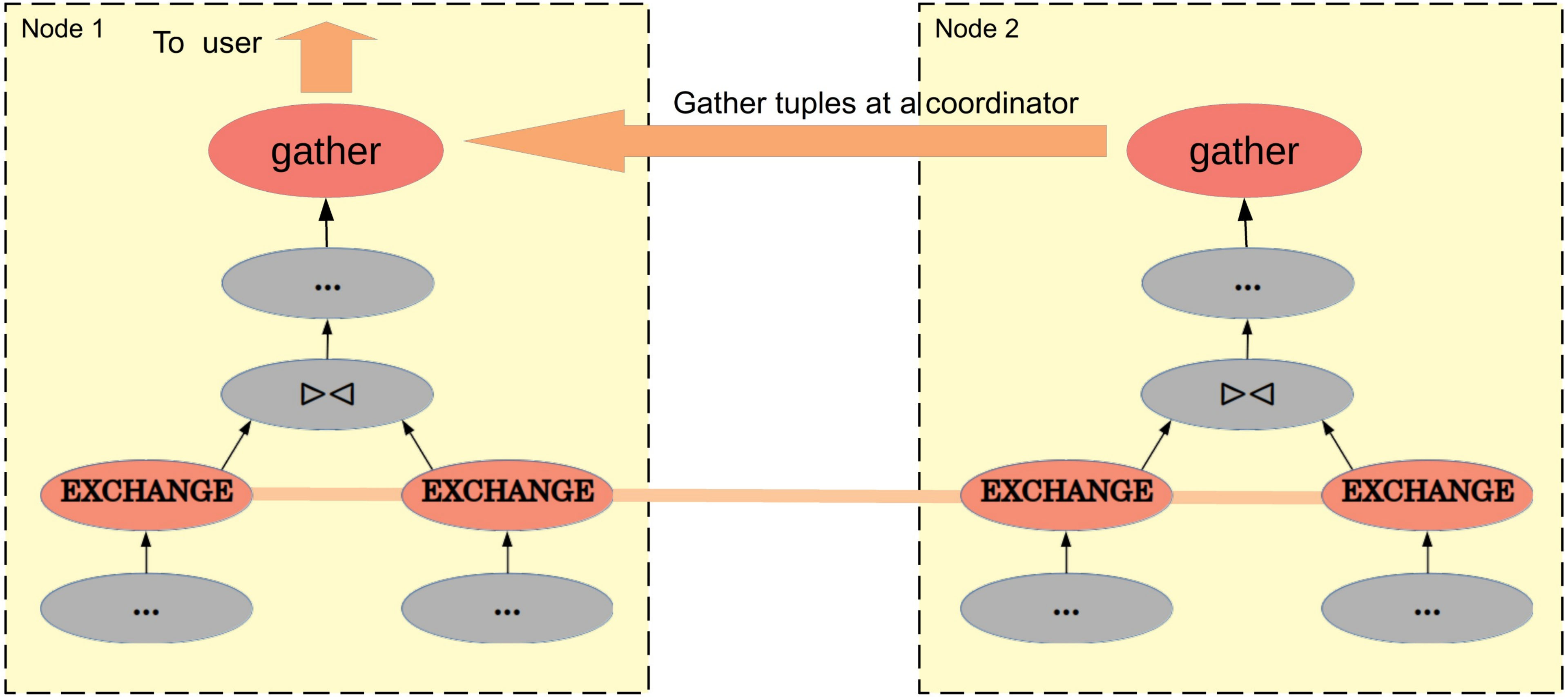
About me

- Ph.D. in Computer Science
- Core Developer in Postgres Professional
- My PostgreSQL Areas:
  - Planner
  - Statistics
  - Access methods
  - WAL

Our sharding way

- Fresh PostgreSQL version
- Useful hackers mailing list patches
- Internal sharding-related patches
- Deploy, management and monitoring infrastructure

Korotkov A., Lepikhov A.
Beyond the pushdowns – distributed query planning and execution // PGConf.EU 2019.


Node 1
To user
Gather tuples at a coordinator
Node 2
gather
...
⋈
EXCHANGE
EXCHANGE
...
...
gather
...
⋈
EXCHANGE
EXCHANGE
...
...

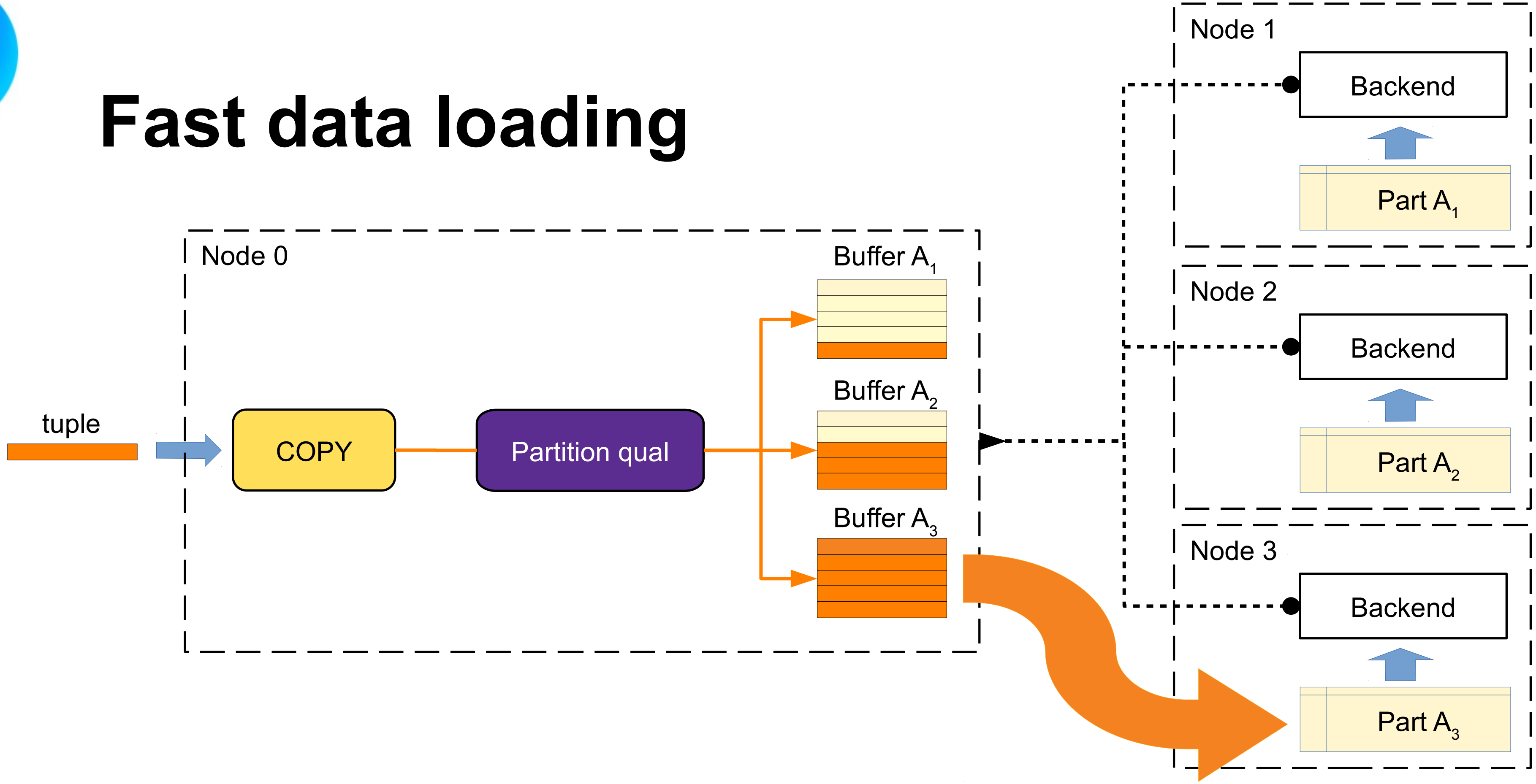
Key ideas

- Reuse existed equipment
  - Partition as a shard
  - FDW as a transport protocol
  - Sharded table == partitioned table with foreign partitions
- Seamless transfer from single instance to a distributed DBMS
- Each PostgreSQL instance can process transactions
- Minimum restrictions on PostgreSQL tools/features

Sharding features

- Fast bulk data loading
- Distributed execution in-parallel
- Global atomcity
- Global snapshot isolation
- Global statistics
- Additional Push-down optimizations
- Resharding

Fast data loading



Buffer Sending Protocol:

1. Execute command: **COPY ..FROM STDIN**
2. Send tuple-by-tuple
3. Send EOF

Fast data loading: benchmarking

Fast COPY FROM Feature:

- Available in the hackers mailing list and commitfest
- More invasive (and faster) version in the Shardman.

Benchmark:

COPY 10 mln tuples into the table:

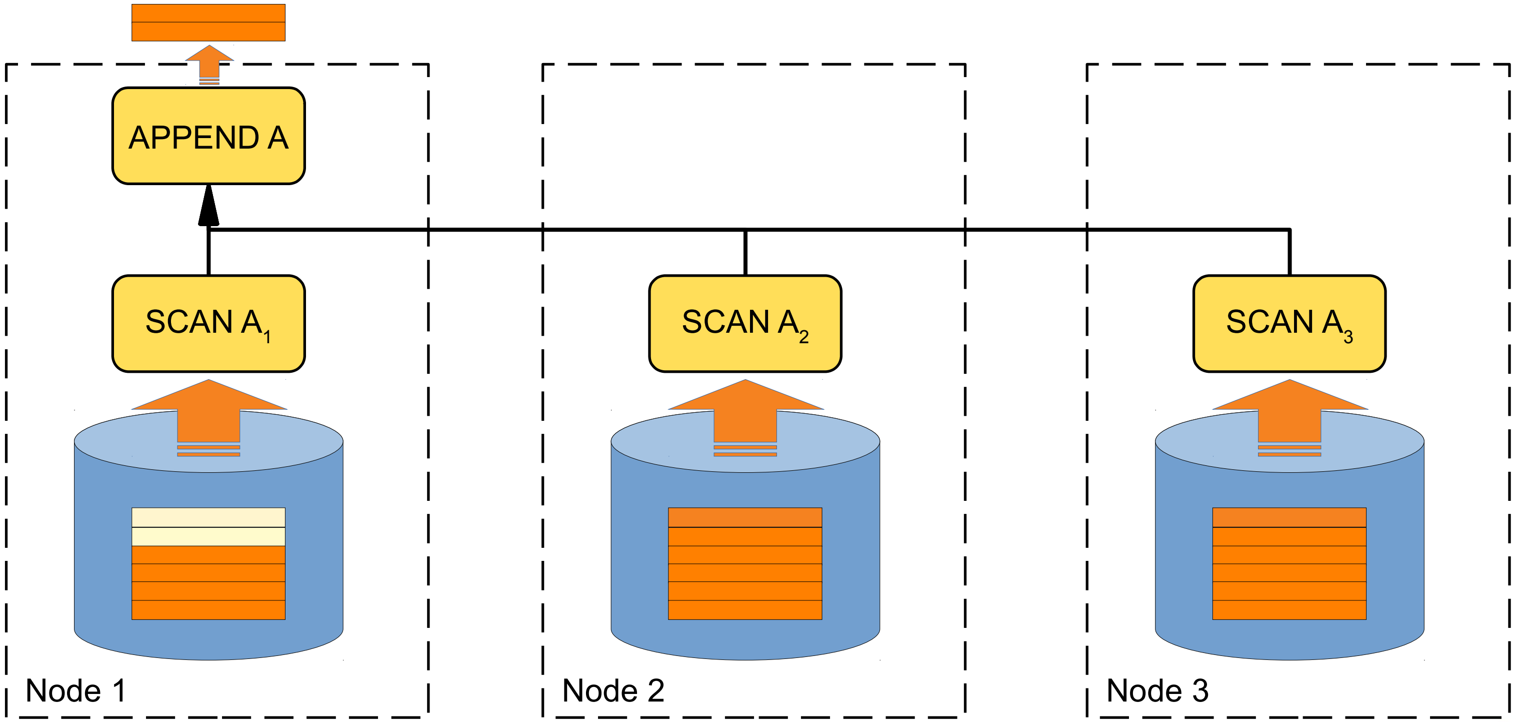
CREATE TABLE test (a int);

PostgreSQL v.13	Fast Copy From (hackers-list)	Fast COPY FROM (Shardman)
14 min 40 sec	35 sec	8 sec

Asynchronous append

The problem

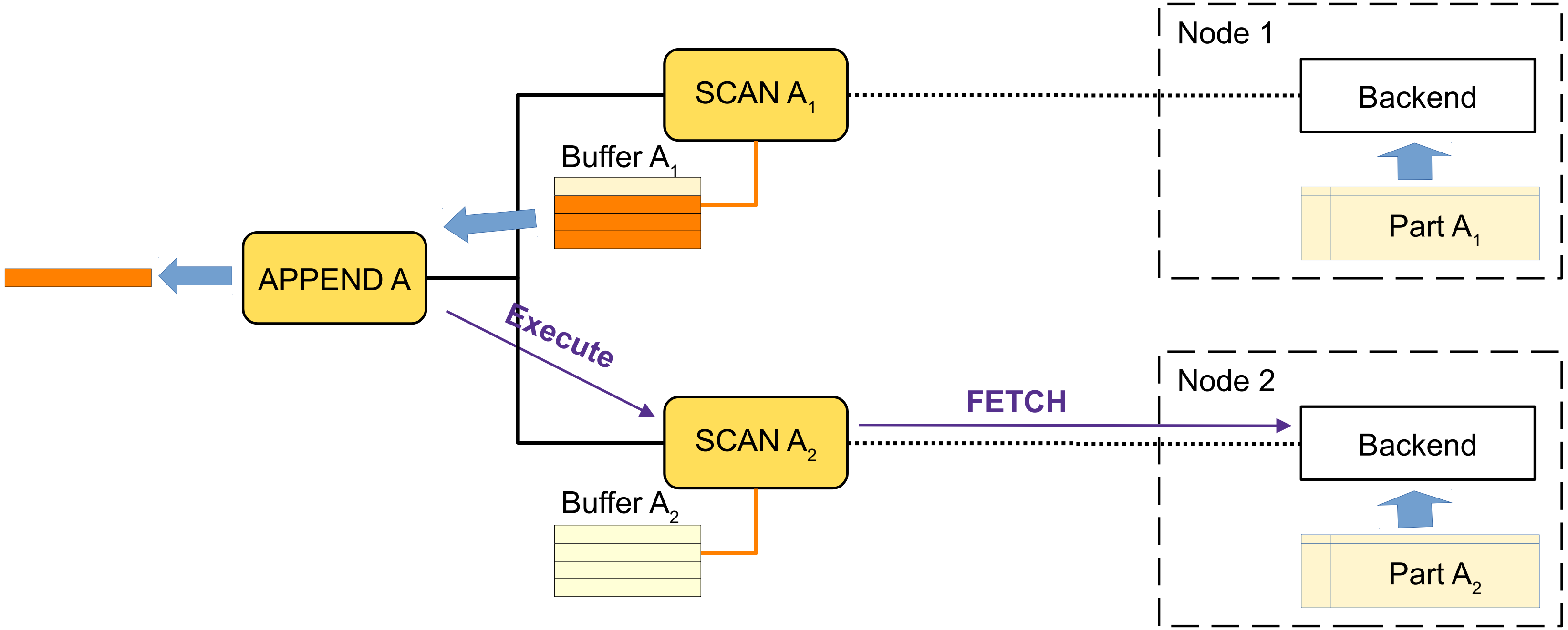
Query: “SELECT * FROM A”



Asynchronous append

Query:

“SELECT * FROM A”



Asynchronous append

explain

```
shardman=# explain (COSTS OFF) SELECT * FROM employees;
                    QUERY PLAN
-----------------------------------------------------------
 Append
   Async subplans: 5
   ->  Async Foreign Scan on employees_0_fdw employees_1
   ->  Async Foreign Scan on employees_1_fdw employees_2
   ->  Async Foreign Scan on employees_2_fdw employees_3
   ->  Async Foreign Scan on employees_4_fdw employees_5
   ->  Async Foreign Scan on employees_5_fdw employees_6
   ->  Seq Scan on employees_3 employees_4
```

Asynchronous append - benchmarking

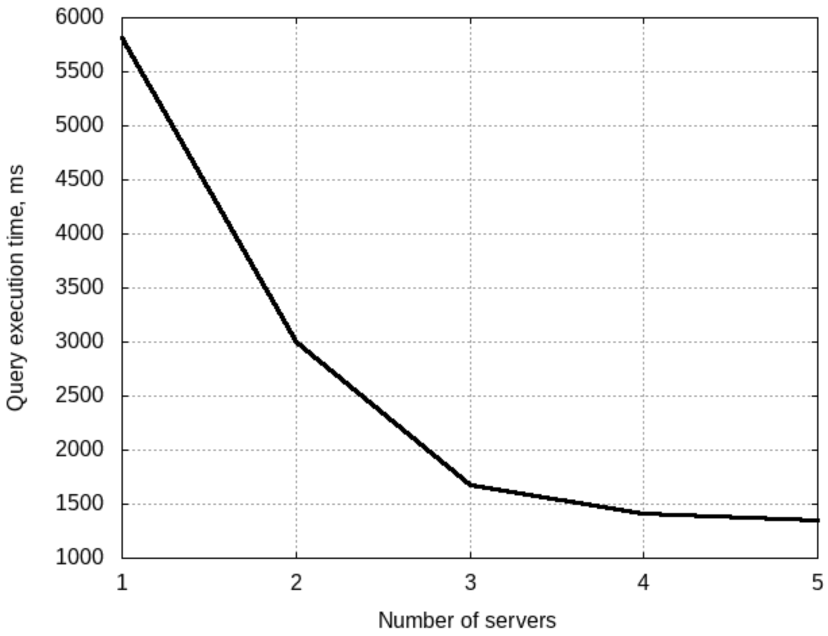
Benchmarking query example:

```
SELECT * FROM partition_0_fdw LIMIT <N>)
UNION ALL
(SELECT * FROM partition_1_fdw LIMIT <N>)
UNION ALL
(SELECT * FROM partition_2_fdw LIMIT <N>)
UNION ALL
(SELECT * FROM partition_3_fdw LIMIT <N>)
UNION ALL
(SELECT * FROM partition_4_fdw LIMIT <N>)
```

Expandability benchmark:

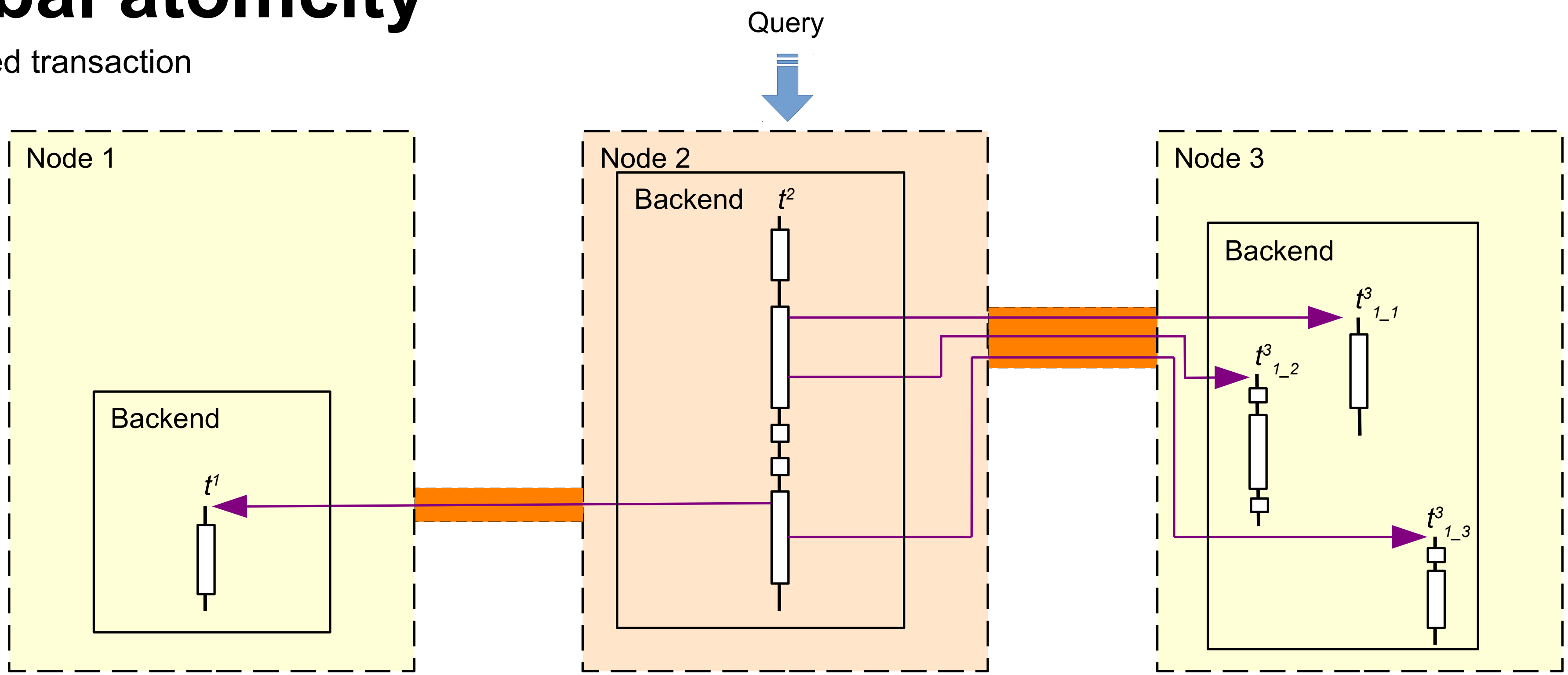
Number of foreign partitions	1	2	3	4	5
Execution time, ms	70	65	69	67	62

Speedup benchmark: (scan 1 mln. tuples from 10 mln. relation)



Global atomcity

Distributed transaction

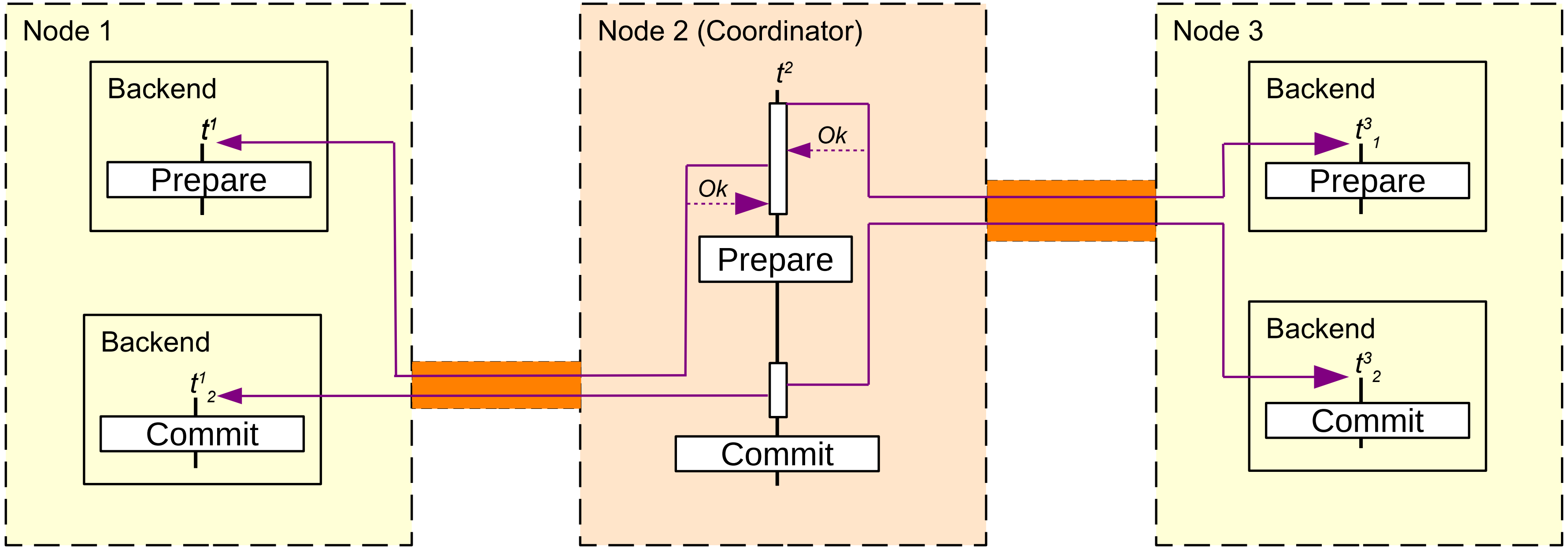


Global atomcity

Distributed commit

Two stages:

- PREPARE on each node. PREPARE on coordinator
- COMMIT on each node. COMMIT on coordinator
- Needs resolving!



Global atomcity

Resolving

GID (Global ID): <node_num>-<xid>

Resolver:

- Get a list of gids of prepared transactions from a node. For each gid:
- If transaction with the **xid** on node **node_num** still active, skip.
- If transaction with the **xid** on node **node_num** isn’t known, rollback prepared**.**
- If node **node_num** have prepared transaction with xid, commit prepared.

Global atomcity

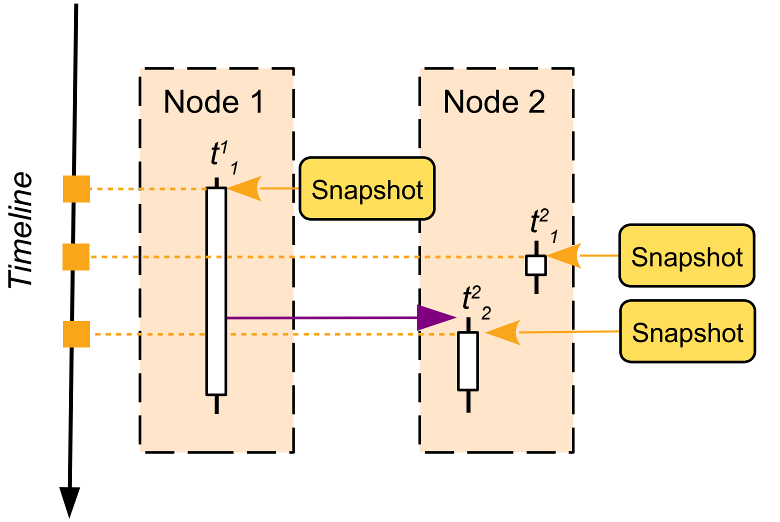
Community

- Global 2PC commit patch can be found here.
- Resolver still not in the hackers mailing list.
- Shardman contains both.

Global snapshot isolation (SI)

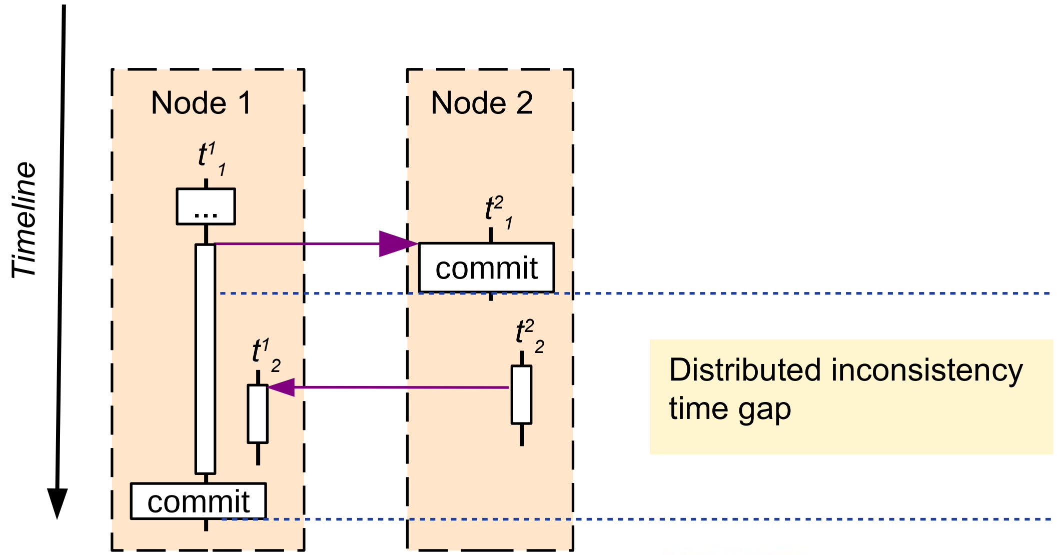
The problem

- REPEATABLE READ



Distributed inconsistency:
Transaction will see future values

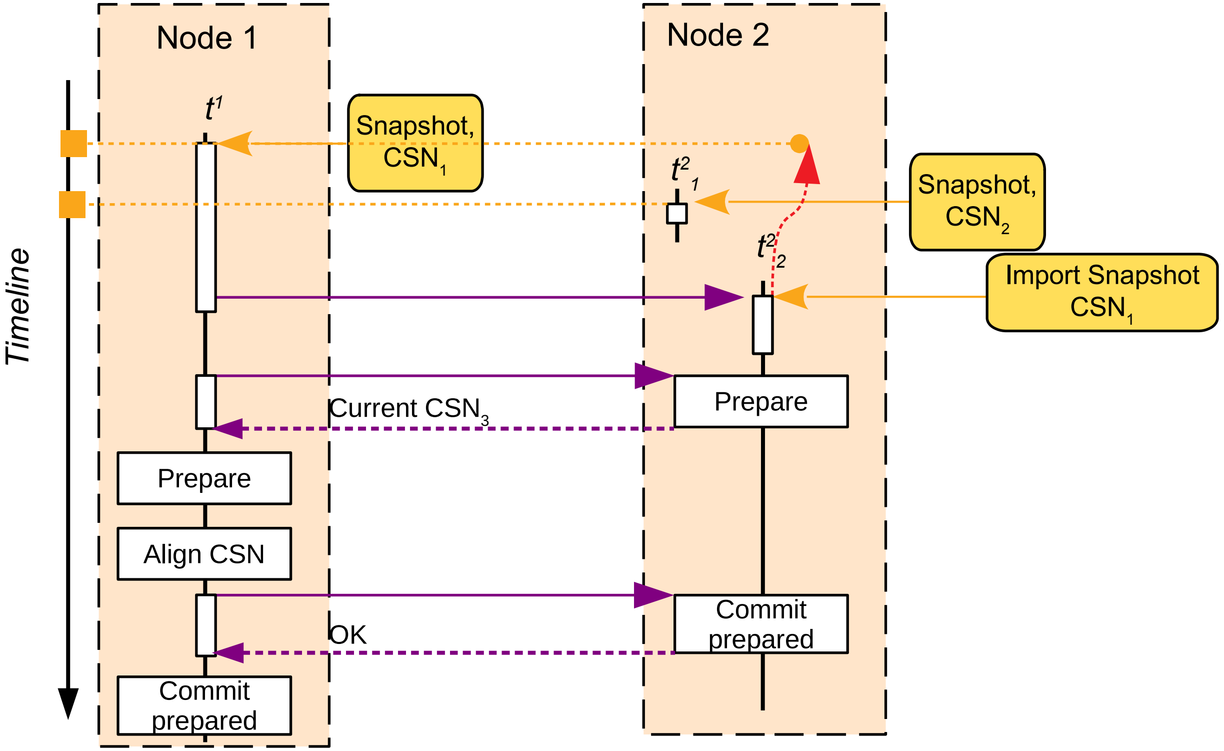
- READ COMMITTED



Distributed inconsistency time gap

Global snapshot isolation

Solution



- **CSN** – **C**ommit **S**equental **N**umber, physical time (may be aligned to a detected skew).
- CSN allocated with any snapshot and is a part of snaphot.
- For several seconds in past is maintained a circular buffer of oldestXmins allowed to shift oldestXmin in the past when backend is importing.

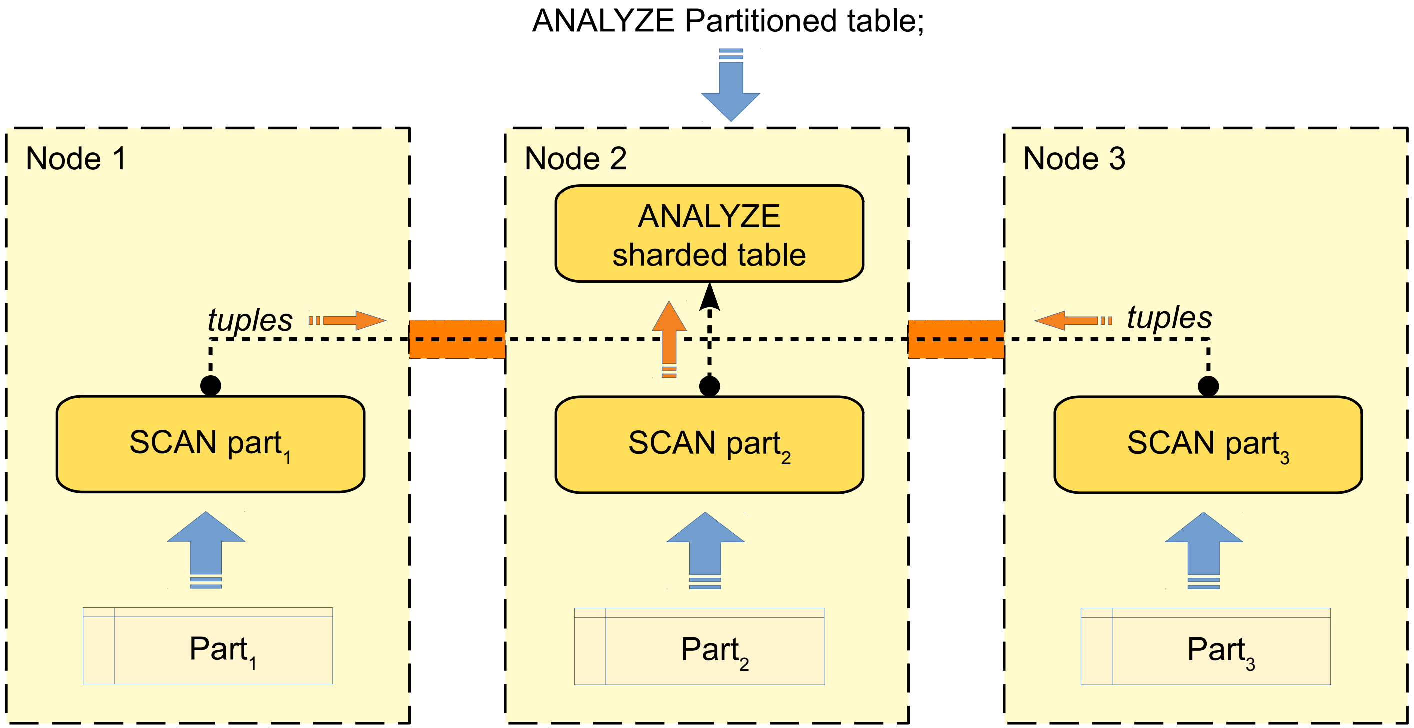
Global snapshot isolation

Community

- Versions and discussion on this feature can be found in the hackers mailing list here, here and here.
- Shardman also contains integration with global commit and resolver features.
- Thanks to HighGo team for many improvements

Global statistics

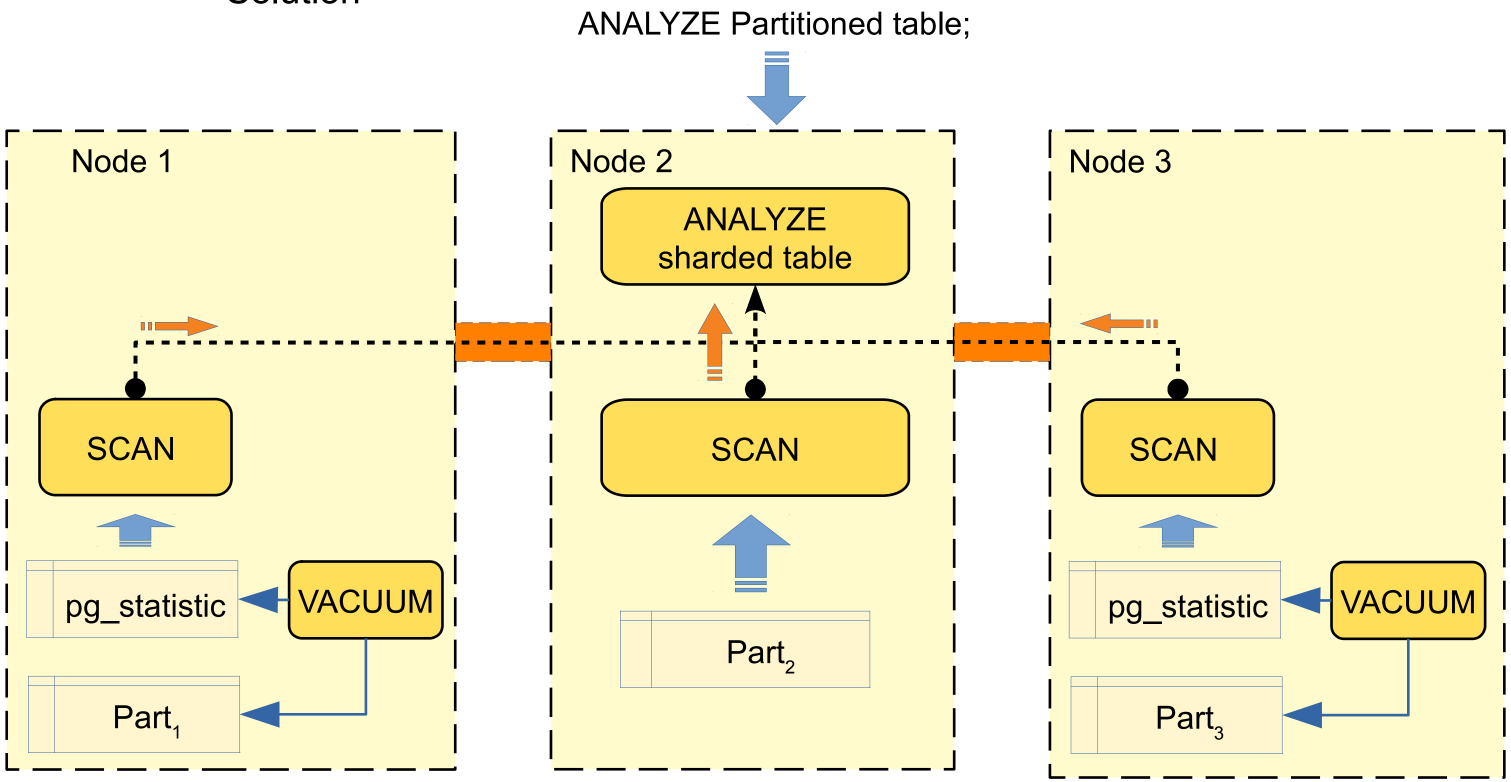
The problem



- To create optimal plan the planner needs fresh statistics.
- ANALYZE of sharded table induces scan and transfer tuples of each foreign partition to the coordinator across network.
- On each instance autovacuum keeps partition statistics fresh by executing ANALYZE locally from time to time.

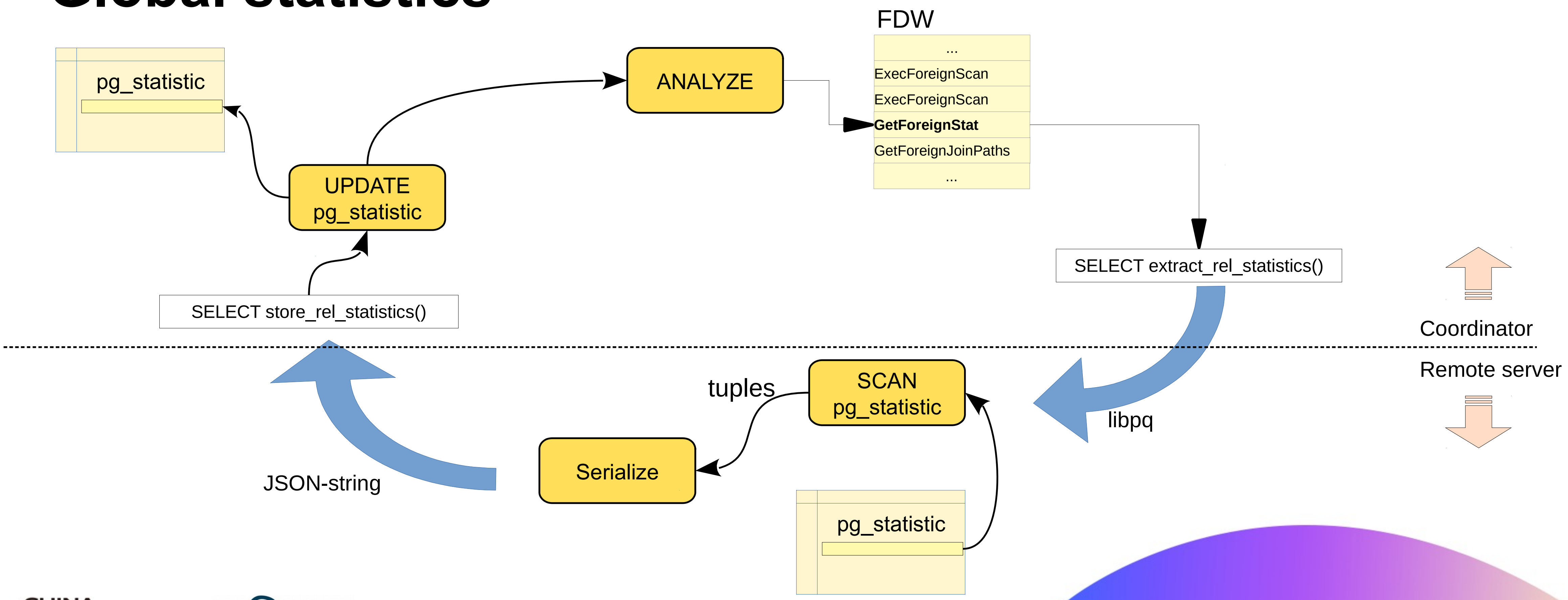
Global statistics

Solution



- *extract_relation_statistics()* - convert statistics tuples to the JSON string.
- *store_relation_statistics()* - parse JSON string to statistics tuples and store into the pg_statistics table.
- To transfer JSON string across network uses FDW.

Global statistics



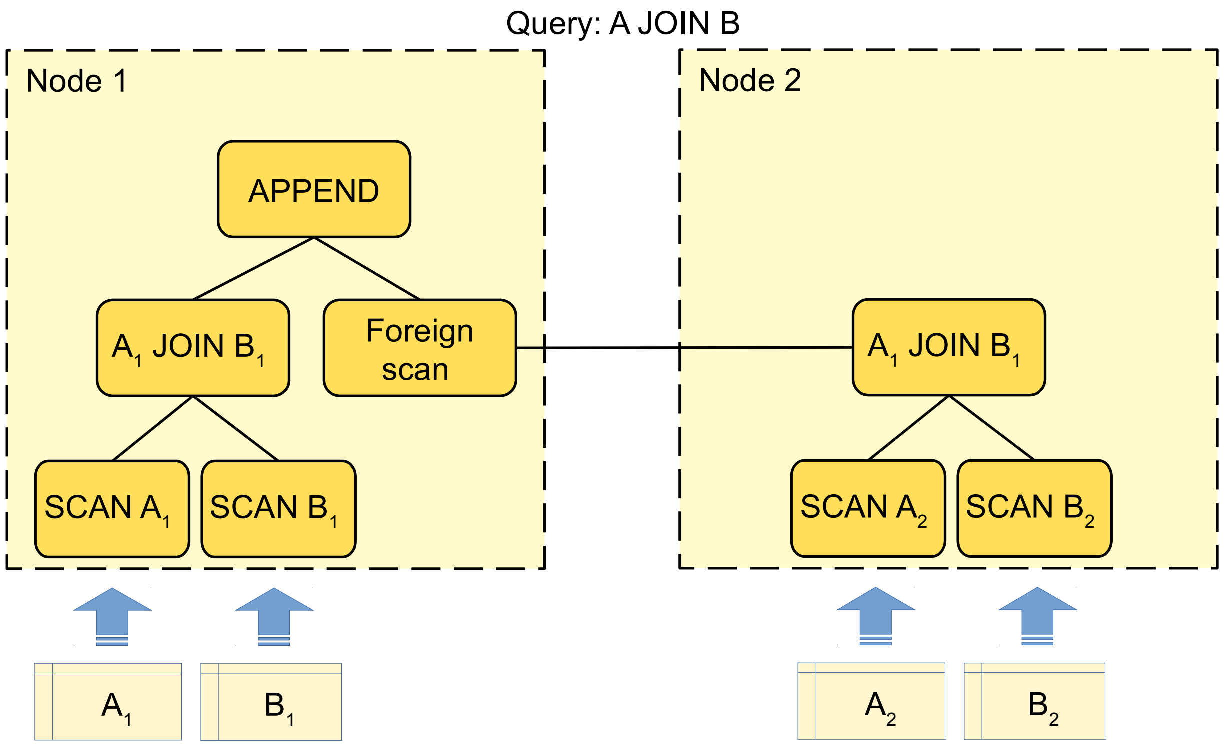
Global statistics

Community

- Actual version of copy statistics functions can be found in the hackers mailing list.
- Shardman also contains integration with the ANALYZE command and changes in autovacuum
- pg_dump & pg_upgrade can utilize this feature also.

Additional push-down optimizations

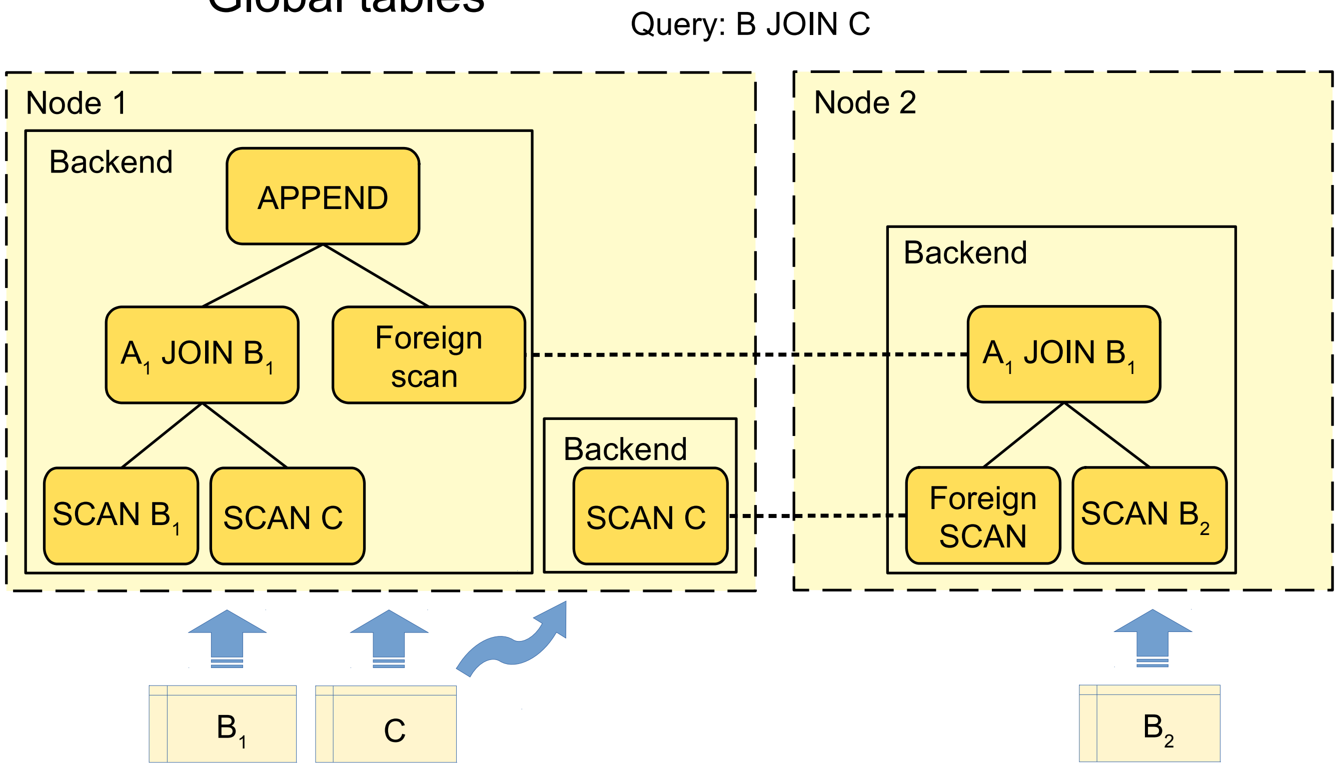
The problem



- Push down join into foreign server if tables partitioned equally.
- Not working with arbitrary partitioning

Additional push-down optimizations

Global tables



- Useful for small relation C and large relation B
- Not breaks push-down machinery if up the query tree exists another join/aggregates.

Additional push-down optimizations

```
EXPLAIN (ANALYZE, COSTS OFF)
SELECT avg(emp_id)
FROM employees e, companies c
WHERE e.cmp_id = c.cmp_id
AND c.cmp_id IN (1,2);
```

Explain

```
Aggregate (actual time=12931.858..12931.858 rows=1 loops=1)
  ->  Hash Join (actual time=4.891..12923.326 rows=20303 loops=1)
        Hash Cond: (e.cmp_id = c.cmp_id)
        ->  Append (actual time=4.285..11691.740 rows=10000000 loops=1)
              Async subplans: 5
              ->  Async Foreign Scan on employees_0_fdw e_1 (rows=1666553 loops=1)
              ->  Async Foreign Scan on employees_1_fdw e_2 (rows=1667504 loops=1)
              ->  Async Foreign Scan on employees_2_fdw e_3 (rows=1665959 loops=1)
              ->  Async Foreign Scan on employees_4_fdw e_5 (rows=1665494 loops=1)
              ->  Async Foreign Scan on employees_5_fdw e_6 (rows=1666482 loops=1)
              ->  Seq Scan on employees_3 e_4 (rows=1668008 loops=1)
        ->  Hash (actual time=0.027..0.028 rows=2 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 9kB
              ->  Index Only Scan using companies_pkey on companies c (rows=2 loops=1)
                    Index Cond: (cmp_id = ANY ('{1,2}'::integer[]))
                    Heap Fetches: 2
 Planning Time: 0.510 ms
 Execution Time: 12939.876 ms
```

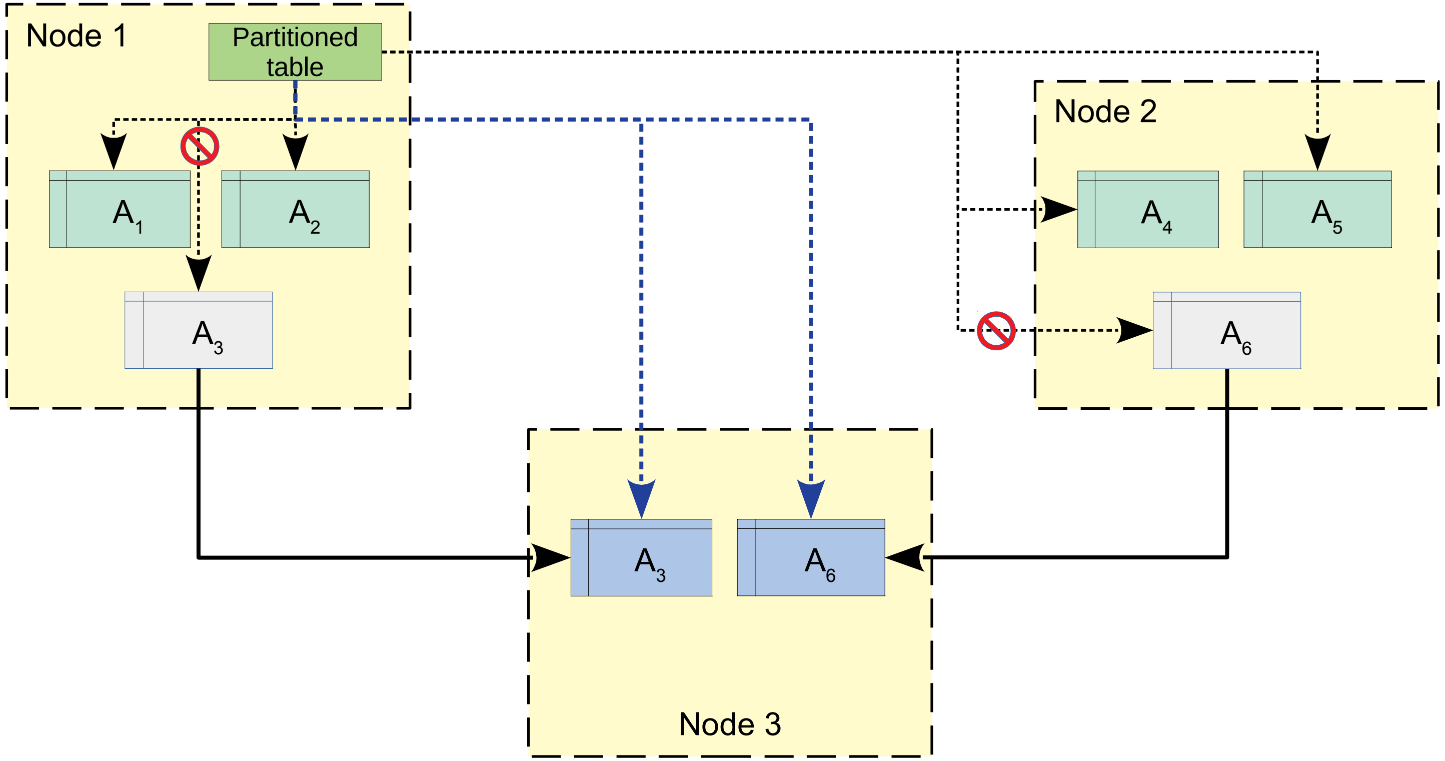
```
Aggregate (actual time=40.101..40.101 rows=1 loops=1)
  ->  Append (actual time=9.597..37.728 rows=20303 loops=1)
        Async subplans: 5
        ->  Async Foreign Scan (rows=3358 loops=1)
              Relations: (employees_0_fdw e_1) INNER JOIN (companies c)
        ->  Async Foreign Scan (rows=3387 loops=1)
              Relations: (employees_1_fdw e_2) INNER JOIN (companies c)
        ->  Async Foreign Scan (rows=3433 loops=1)
              Relations: (employees_2_fdw e_3) INNER JOIN (companies c)
        ->  Async Foreign Scan (rows=3326 loops=1)
              Relations: (employees_4_fdw e_5) INNER JOIN (companies c)
        ->  Async Foreign Scan (rows=3404 loops=1)
              Relations: (employees_5_fdw e_6) INNER JOIN (companies c)
        ->  Nested Loop (rows=3395 loops=1)
              ->  Index Only Scan using companies_pkey on companies c (rows=2 loops=1)
                    Index Cond: (cmp_id = ANY ('{1,2}'::integer[]))
                    Heap Fetches: 2
              ->  Bitmap Heap Scan on employees_3 e_4 (rows=1698 loops=2)
                    Recheck Cond: (cmp_id = c.cmp_id)
                    Heap Blocks: exact=3271
                    ->  Bitmap Index Scan on employees_3_cmp_id_idx (rows=1698 loops=2)
                          Index Cond: (cmp_id = c.cmp_id)
 Planning Time: 1.063 ms
 Execution Time: 50.747 ms
```

Additional push-down optimizations

Community

- Based on commitfest patch on asymmetric partitionwise join.
- Global tables still not exists in the hackers mailing list.
- Implemented in the Shardman.

Resharding



- Based on logical replication
- Seamless migration of partitions (A_3 and A_6 on the slide)
- Quick switch with detach old partition (A_3 on node 1 and A_6 on node 2) and attach the new.

CONTACT

THANKS

a.lepikhov@postgrespro.ru

@avlepikhov

CHINA POSTGRESQL ASSOCIATION | PCC POSTGRESCONF CN 2020 | PGConf.Asia