Scaling with PostgreSQL 9.6 and Postgres-XL

Mason Sharp
mason.sharp@gmail.com
Huawei

whoami

- Engineer at Huawei (USA based)
- Co-organizer of NYC PostgreSQL User Group
- PostgreSQL-based database clusters
  - GridSQL
  - Postgres-XC
  - Postgres-XL

Agenda

- PostgreSQL 9.6 Scaling Features
- Scaling Solutions
- Use Cases
- Postgres-XL

BTW- PGConf US 2017 in New York City

- Join us!
- Submit a talk

“We need to scale”

“We need replication”

What are your requirements?

What are your requirements?

- Is the workload for OLTP, OLAP or is it mixed?
- What percentage of transactions are read-only vs. write?
- If using replicas, may they lag behind? Or must they be synchronous?
- If sharding across multiple instances:
  - Will you have distributed transactions?
  - Do you require distributed consistency?
  - What isolation level is being used?
- Will there be a node at a remote location?
  - What is acceptable latency?
  - What is acceptable in a network partition?
- Do you require using only PostgreSQL binaries and extensions, or are can you utilize other projects?

First, Quick Performance Ideas

- Put WAL logs (pg_xlog directory) on a separate storage device
- Hundreds of connections?
  - Use pgbouncer connection pooler
- postgresql.conf
  - Increase shared_buffers
  - effective_cache_size
  - Checkpoints
    - checkpoint_completion_target = 0.9
    - Increase checkpoint_timeout if longer recovery time is ok
  - Increase work_mem
  - Increase maintance_work_mem

Can new PostgreSQL 9.6 features help?

PostgreSQL 9.6 Replication

- **New option in 9.6:** synchronous_commit = remote_apply
  - remote_write: write, but no fsync
  - on: write and fsync
  - remote_apply: write, fsync, and wait until WAL is applied to buffer
- Before this, even with synchronous_commit = on, it was possible that after writing to the database and reading from a replica that the data might not yet have be applied..
- Synchronous replication (on) may reduce write transaction throughput. Using remote_apply potentially more so
- If it is acceptable to have stale reads from replicas, and it is ok to lose some data and you need more write throughput, consider synchronous_commit off. Can set per transaction.

PostgreSQL 9.6 Replication

- Change application so that write transactions use one host, and read-only transactions use a replicas
  - or use pgpool-II
    - Adds more complexity and an extra hop though
    - But can help with availability

PostgreSQL 9.6 Parallel Query

- **New in 9.6**: Parallel Query
- Prior to 9.6, if you had a 32 core system, needed to run one large query, and you were the only user on the system, that query would still just utilize one single core, with 31 cores largely idle
- Prior to 9.6, one work-around for this would be to configure a multi-node Postgres-XL “cluster” on the same server, which would utilize the other cores
- PostgreSQL 9.6 adds parallel operations
  - Sequential scans
  - Joins
  - Aggregates
- Off by default. Set max_parallel_workers_per_gather > 0 to enable
  - max_worker_processes: default is 8, may need to increase
  - force_parallel_mode: off (default) | on | regress
  - parallel_setup_cost: 1000 default
  - parallel_tuple_cost: 0.1 default cost for transferring to another process

PostgreSQL 9.6 pglogical

- contrib module
- In 9.6, no patching required to add in BDR
  - No global sequences
- pglogical extension for logical replication
  - Physical (block level) streaming replication replicates everything
  - pglogical's logical replication permits selective replication for subsets of data for example

Bidirectional Replication (BDR)

- 2nd Quadrant's BDR Bidirectional Replication provides multi-master capability to PostgreSQL. Ability to write at multiple locations. Uses asynchronous logical replication
- Caveats:
  - 1.0 Release
  - DDL requires global write lock
  - Asynchronous must be acceptable
  - Conflict resolution after commit time (last one wins by default)
    - Carefully read more about potential issues: http://bdr-project.org/docs/stable/conflicts-types.html

May be best to try to avoid conflicts if possible. Can try to design schema so that each location "owns" subset of data. Network partitions mean locations can still read and write their own data, and read stale remote data

PostgreSQL 9.6 FDW improvements

- postgres_fdw is a Foreign Data Wrapper to access remote PostgreSQL instances from within PostgreSQL
- In 9.6 postgres_fdw now supports pushing down remote joins, sorts, UPDATEs and DELETEs
- Progress towards federating PostgreSQL servers
- Potential consistency issues
- No direct node-to-node row shipping
- No replicated table handling
- Can be ok for simple queries- SELECT COUNT(*) FROM bigtable;

Other Solutions: PL/Proxy

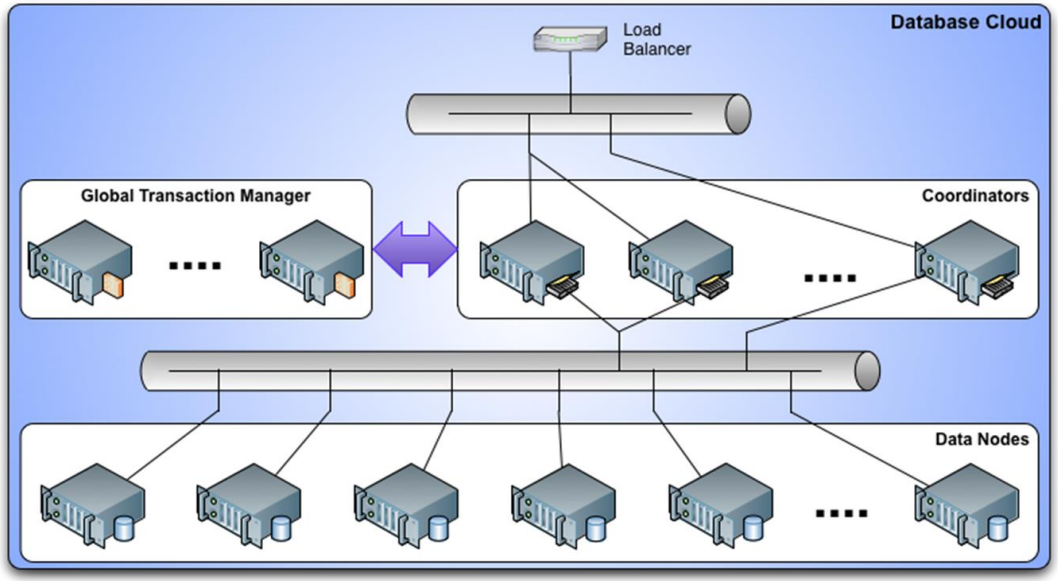
- Created by Skype, used at big sites like MeetMe
- Proxy between applications and actual data
- PL = Procedural Language
- Must write two stored functions for every database access
  - Function on shard node for actual data access
  - Top level proxy function determines which shard to use (or ANY or ALL)
- Hash typically used for sharding the data
- Can also write manual aggregates
- Uses auto-commit mode

Solutions: Custom Schema-level Sharding

- Popularized by Instagram
  - http://media.postgresql.org/sfpug/Instagram_sfpug.pdf
- Use schema for shards
- Create many logical shards, map to physical nodes
- More shards than nodes = flexibility to move shards with growth
- Custom data access functions in application layer to choose appropriate shards, then map shard to node (Python)

Solutions – Postgres-XL

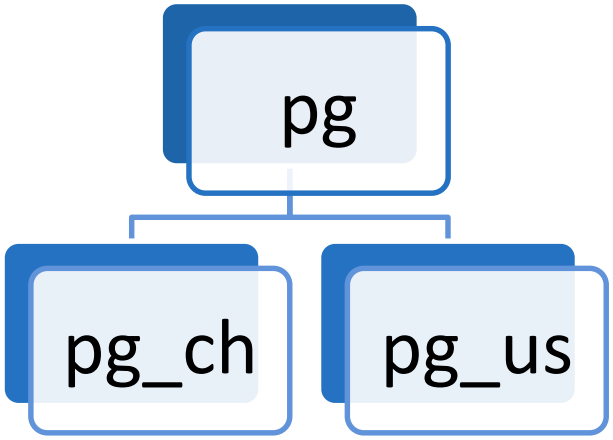
- Scale-out RDBMS
- MPP for OLAP
- OLTP write and read scalability
- Custer-wide ACID properties
- Can also act as distributed key-value store
- Currently based on PostgreSQL 9.5



Postgres-XL

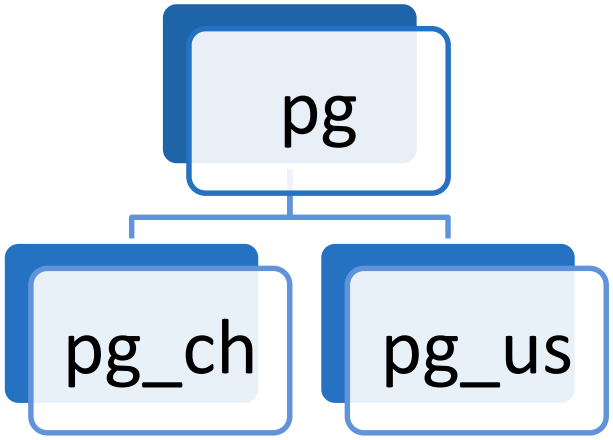
More About postgres_fdw

postgres_fw example




```
CREATE EXTENSION "postgres_fdw";

CREATE SERVER pg_ch
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host '127.0.0.1', port '35433', dbname 'test1');

CREATE USER MAPPING FOR msharp
    SERVER pg_ch
    OPTIONS (user 'msharp');

CREATE SERVER pg_us
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host '127.0.0.1', port '35434', dbname 'test1');

CREATE USER MAPPING FOR msharp
    SERVER pg_us
    OPTIONS (user 'msharp');
```

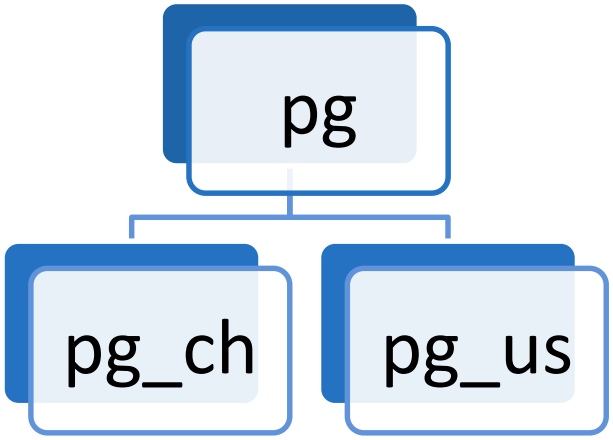
postgres_fdw example



- On all 3 servers, execute:

```
CREATE TABLE invoice
(invoice_id INT,
 invoice_amount NUMERIC(12,2),
 country CHAR(2));
```

postgres_fdw example

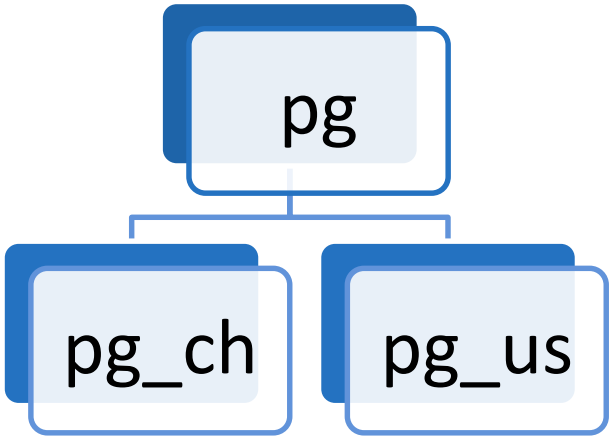


- Supports table inheritance!

```
CREATE FOREIGN TABLE invoice_ch (
 CONSTRAINT cons_inv_china
    CHECK (country = 'CH')
)
INHERITS (invoice)
SERVER pg_ch OPTIONS (table_name 'invoice');
```

```
CREATE FOREIGN TABLE invoice_us (
 CONSTRAINT cons_inv_us
    CHECK (country = 'US')
)
INHERITS (invoice)
SERVER pg_us OPTIONS (table_name 'invoice');
```

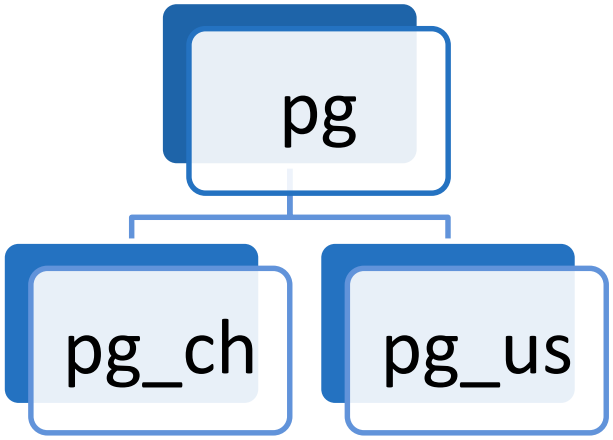
postgres_fdw example



- From pg, execute

```
INSERT INTO invoice_ch
VALUES (1, 100, 'CH');

INSERT INTO invoice_ch
VALUES (2, 150, 'CH');

INSERT INTO invoice_us
VALUES (3, 120, 'US');

INSERT INTO invoice_us
VALUES (4, 75, 'US');
```

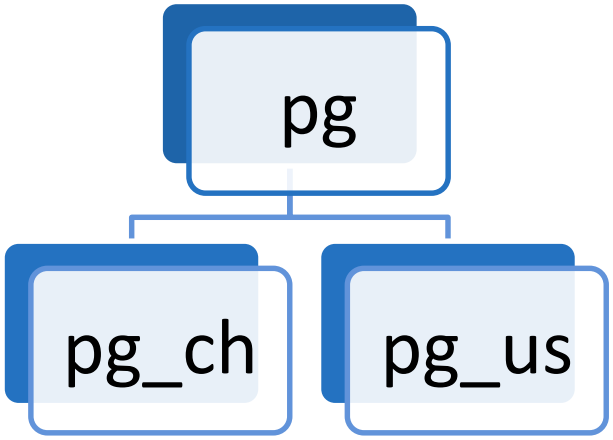
postgres_fdw example



- From pg, execute

```
EXPLAIN SELECT * FROM invoice
WHERE invoice_id = 1;

Append  (cost=0.00..347.89 rows=39 width=31)
  ->  Seq Scan on invoice
        Filter: (invoice_id = 1)
  -> Foreign Scan on invoice_ch
  -> Foreign Scan on invoice_us
```

postgres_fdw example



- From pg, execute

```
EXPLAIN SELECT * FROM invoice
WHERE invoice_id = 1;

Append  (cost=0.00..347.89 rows=39 width=31)
  ->  Seq Scan on invoice
        Filter: (invoice_id = 1)
  -> Foreign Scan on invoice_ch
  -> Foreign Scan on invoice_us

EXPLAIN SELECT * FROM invoice
WHERE invoice_id = 1 AND country = 'CH';

 Append  (cost=0.00..265.21 rows=2 width=22)
  ->  Seq Scan on invoice
        Filter: ((invoice_id = 1) AND (country = 'CH'::bpchar))
  -> Foreign Scan on invoice_ch
```

Avoids using invoice_us

Postgres_fdw Comments

- Will continue to get better with each version
  - Aggregate push down coming soon!
- Join push-down exists, but no notion of “replicated table” to push down across all nodes for joining with static lookup tables
  - Would make usable for star schemas with replicated dimensions
- No direct node–node shipping for joins, some joins cannot be parallelized and pushed-down, must take place at originating PostgreSQL instance
- No distributed consistency

Postgres-XL can do all of the above

It may be difficult to build a scalability solution relying on postgres_fdw, unless schema can be partitioned in such a way to avoid distributed joins altogether, and any multi-node federated queries are simple

Use Cases

OLTP + some reporting

- Offload reporting to a separate server
- Use streaming replication

Read-Only / Read-Mainly Workload

- Use streaming replication
- Use application-level defined read-only connections or something like pgpool-II

- Low cache rates on replicas?

  Consider using pglogical and having replicas contain a subset of the data
  - Asynchronous
  - Applications need data location awareness

OLTP – Write-Heavy Workload

- Use custom sharding such as schema-level
- PL/Proxy
- Postgres-XL

Strong Consistency Required

- Postgres-XL

Geo-distributed (non-partitioned)

- Remote read-only slave(s)
- Use Bidirectional Replication (BDR) extension
  - See earlier caveats

OLAP / Reporting

PostgreSQL

- Use PostgreSQL 9.6 Parallel Query
- Use table inheritance to create subtables for ranges of data
  - Typically by Date
  - Optimizer may plan to avoid accessing large subsets of data
- Use partial indexes
  - CREATE INDEX country_ch_idx ON orders (country)
    WHERE country = ‘CH’

Extensions

- For a few simple queries, postgres_fdw or PL/Proxy could be used

Other

- Postgres-XL (The PostgreSQL License)
- Greenplum (Apache License)

Mixed Workload – HTAP
Hybrid Transactional Analytical Processing

- Postgres-XL

More About

Postgres-XL

- Branch? Fork?
- **Try to stay close to PostgreSQL**
- Currently 9.5
  - Does not consume transaction id for single node read
- 9.6 just starting to be merged in
  - Will be able to use parallelism within nodes and across nodes
- 2ndQuadrant active with further development

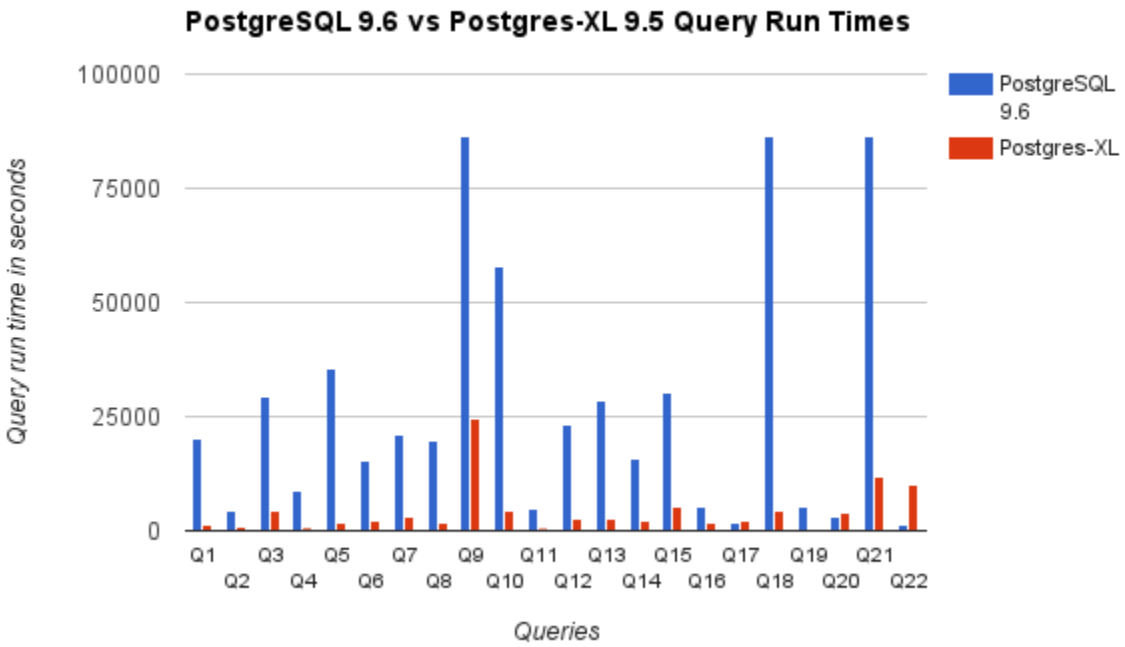
Postgres-X* Landscape

- Postgres-XL based on Postgres-XC
  - Data node to data node communication for performance
- Postgres-X2 based on PG 9.3, continuation of XC
- Postgres-XC fork called Postgres-XZ created by Tencent

Postgres-XL Missing Features

- Built-in High Availability
  - Use external like Corosync/Pacemaker
  - Area for future improvement
- “Easy” to re-shard / add nodes
  - Some pieces there
  - Tables locked during redistribution
  - Can however “pre-shard” multiple nodes on the same server or VM
- Some FK and UNIQUE constraints
- Recursive CTEs (WITH)

3 TB Postgres-XL 9.5



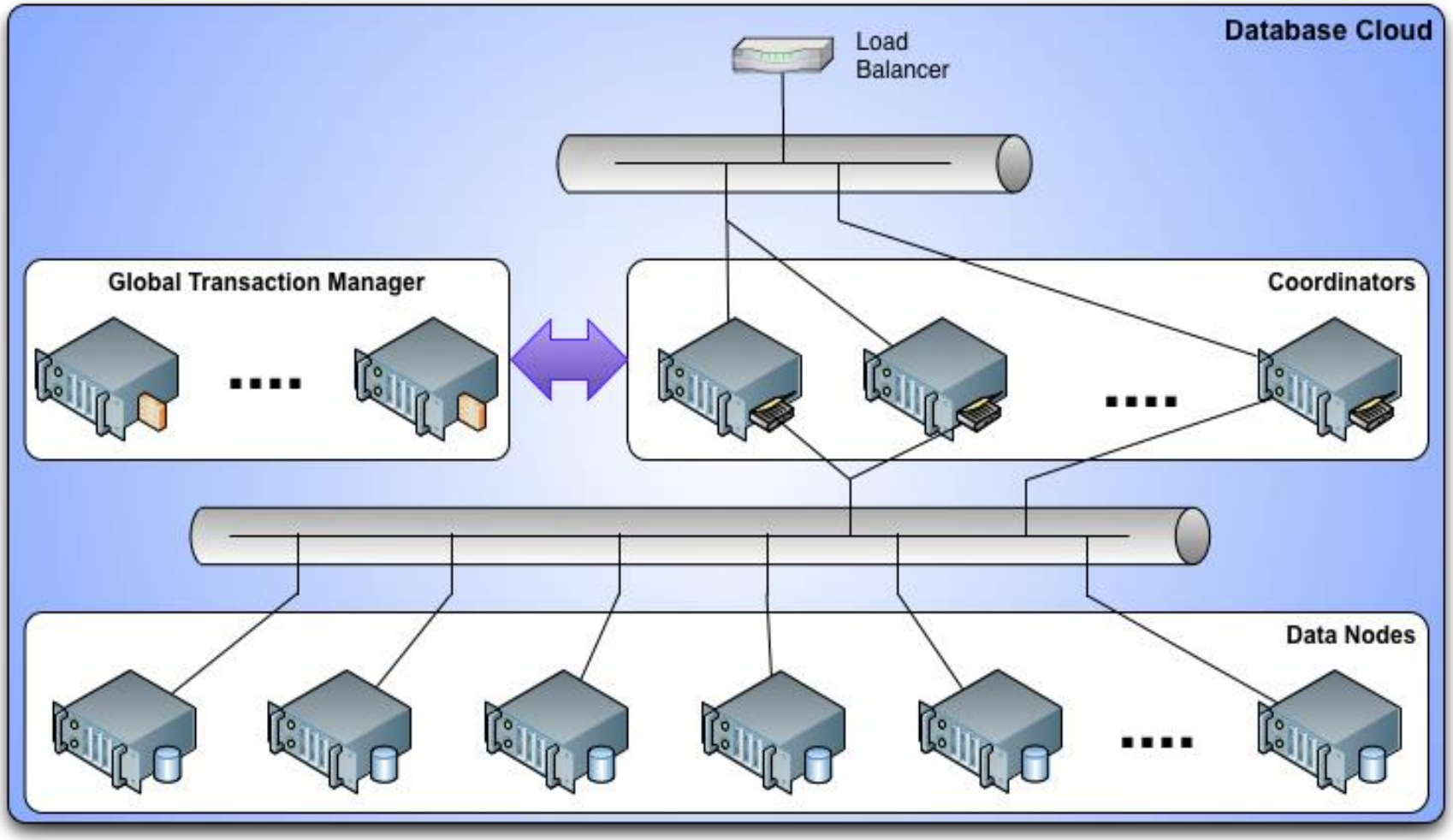
Source: blog.2ndquadrant.com/benchmarking-postgres-xl

Website and Code

- postgres-xl.org
- git://git.postgresql.org/git/postgres-xl.git
- Docs
  - files.postgres-xl.org/documentation/index.html

Roadmap

- 9.6 Merge
- Global Transaction Manager Improvements
  - Including making it optional when not needed
- Make XL a more robust analytical platform
  - Distributed Foreign Data Wrappers
    - Columnar compressed FDW
    - SQL on HDFS
- High Availability
- Elasticity
  - Break 1-1 mapping of shard to node

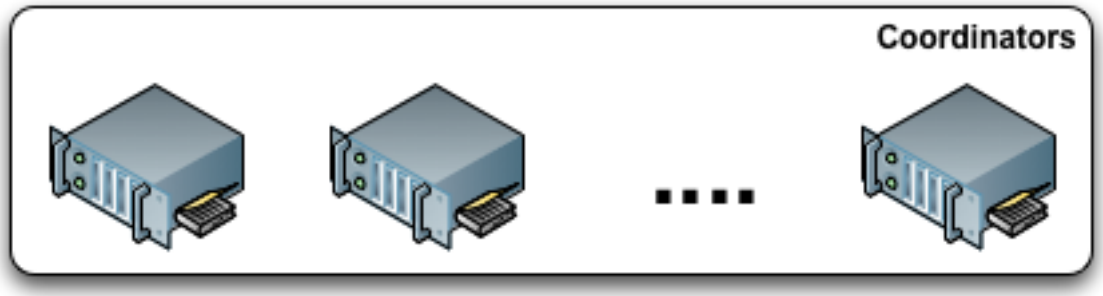

Database Cloud
Load
Balancer
Global Transaction Manager
Coordinators
Data Nodes

Postgres-XL Architecture

- Coordinators
  - Connection point for applications
  - Parsing and planning of queries
- Data Nodes
  - Actual data storage
  - Local execution of queries
  - Direct Data Node <-> Data Node communication and row shipping
- Global Transaction Manager (GTM)
  - Provides a consistent view to transactions
  - GTM Proxy to increase performance

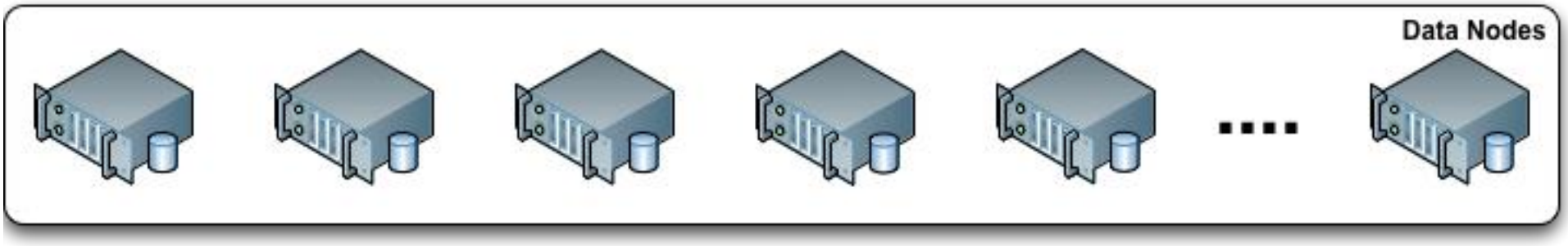
Coordinators

- Handles network connectivity to the client
- Parse and plan statements
- Perform final query processing of intermediate result sets
- Manages two-phase commit
- Stores global catalog information



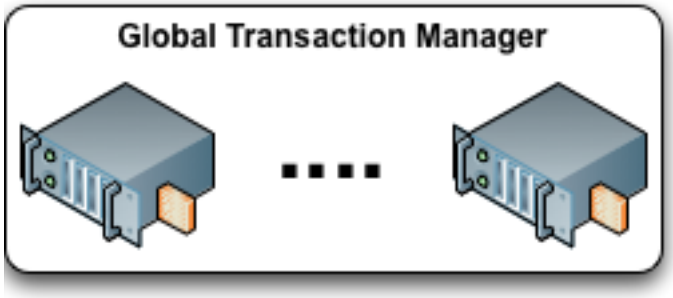
Data Nodes

- Stores tables and indexes
- Only coordinators connects to data nodes
- Executes queries from the coordinators
- Communicates with peer data nodes for distributed joins



Global Transaction Manager (GTM)

- Handles necessary MVCC tasks
  - Transaction IDs
  - Snapshots
- Manages cluster wide values
  - Timestamps
  - Sequences
- GTM Standby can be configured



Thanks!

Q & A

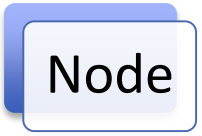
Backup Slides

Multiversion Concurrency Control (MVCC)

MVCC

- Readers do not block writers
- Writers do not block readers
- Transaction Ids (XIDs)
  - Every transaction gets an ID
- Snapshots contain a list of running XIDs

MVCC – Regular PostgreSQL



Example:

T1 Begin... INSERT…

T2 Begin; INSERT...; Commit;

T3 Begin... INSERT…

T4 Begin; SELECT

- T4's snapshot contains T1 and T3
  - T2 already committed
  - T4 can see T2's commits, but not T1's nor T3's

MVCC – Multiple Node Issues

Example:

T1 Begin...

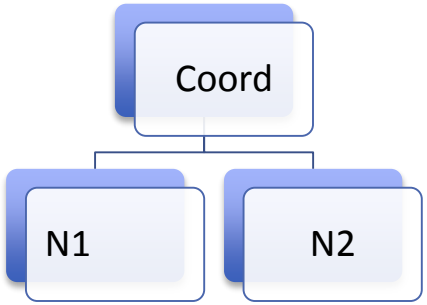
T2 Begin; INSERT...; Commit;

T3 Begin...

T4 Begin; SELECT

N1,N2

N2,N1

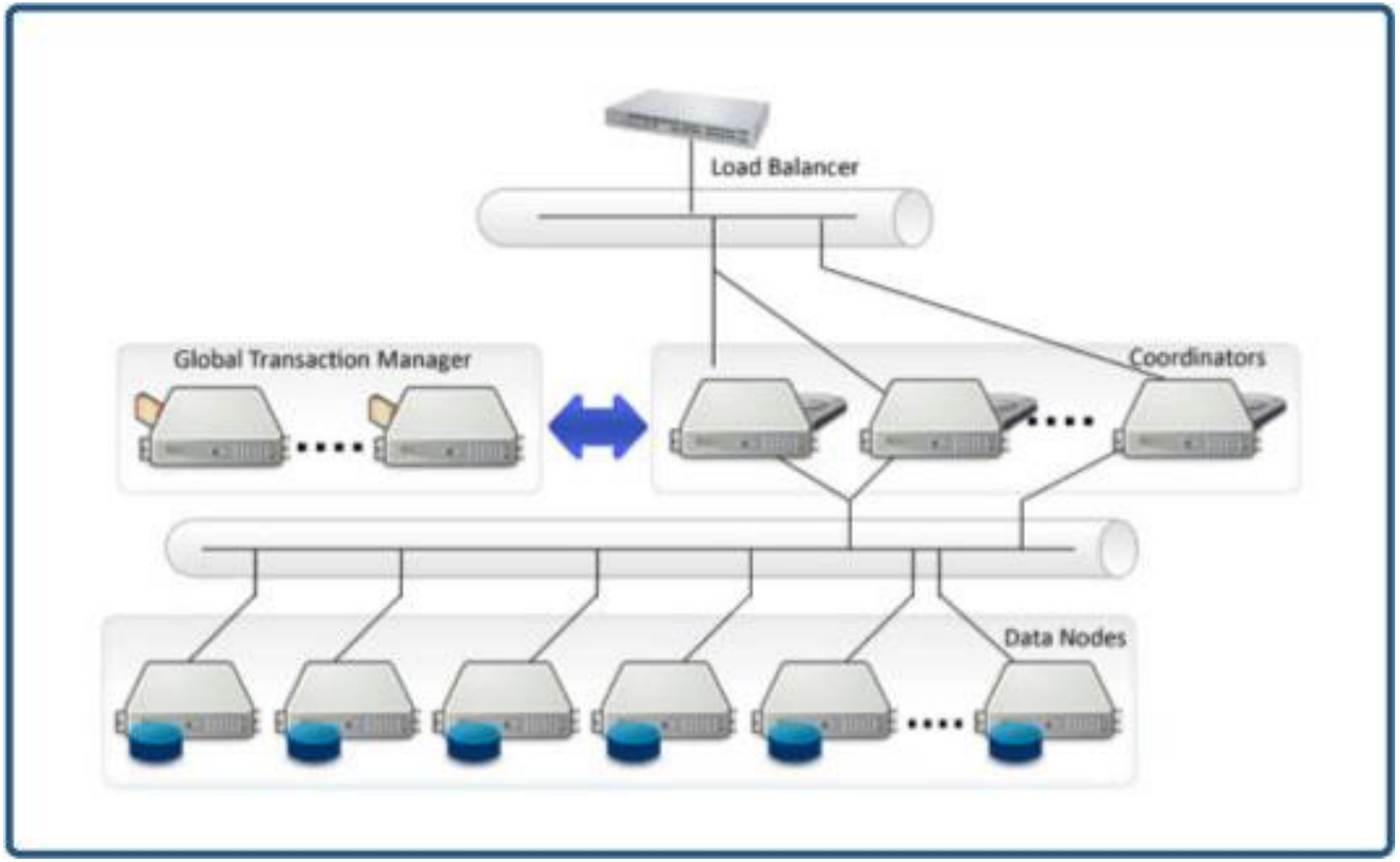


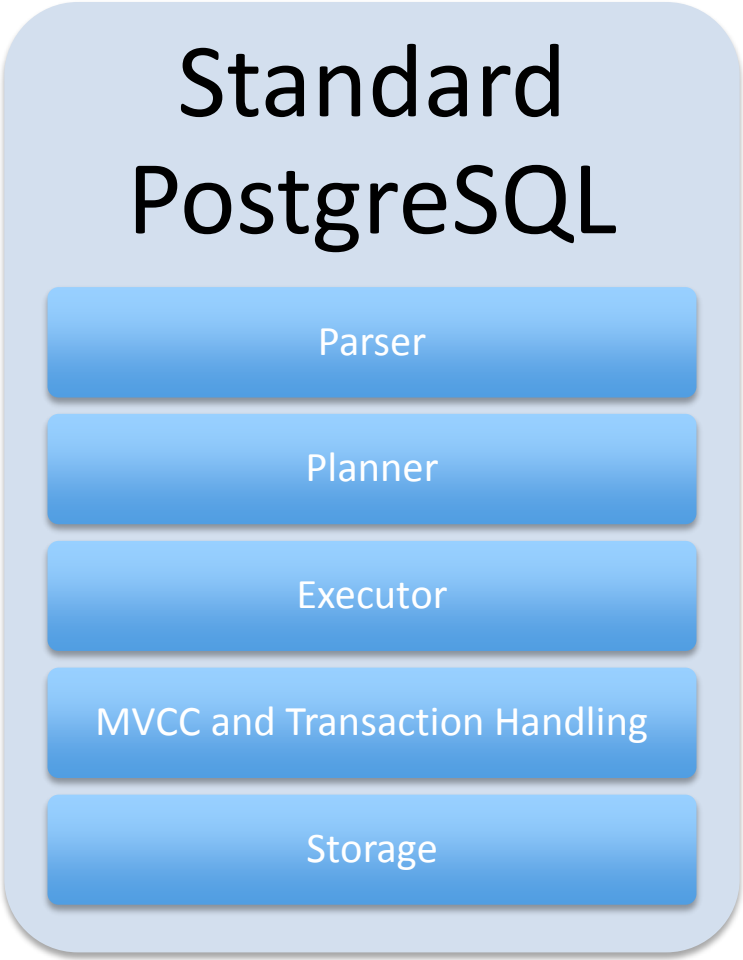
- Node 1: T2 Commit, T4 SELECT
- Node 2: T4 SELECT, T2 Commit
- T4's SELECT statement returns inconsistent data
  - Includes data from Node1, but not Node2.

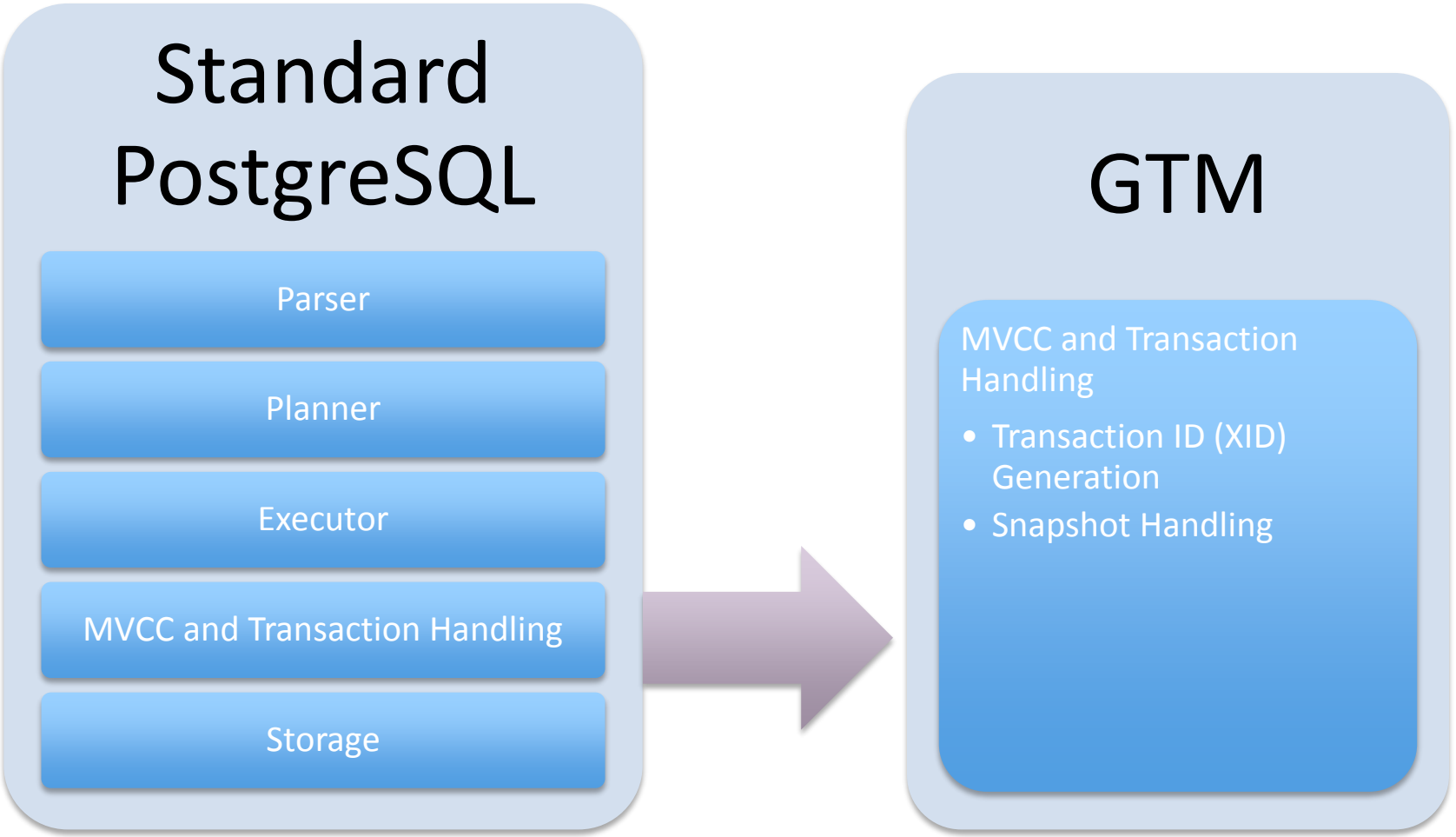
MVCC – Multiple Node Issues

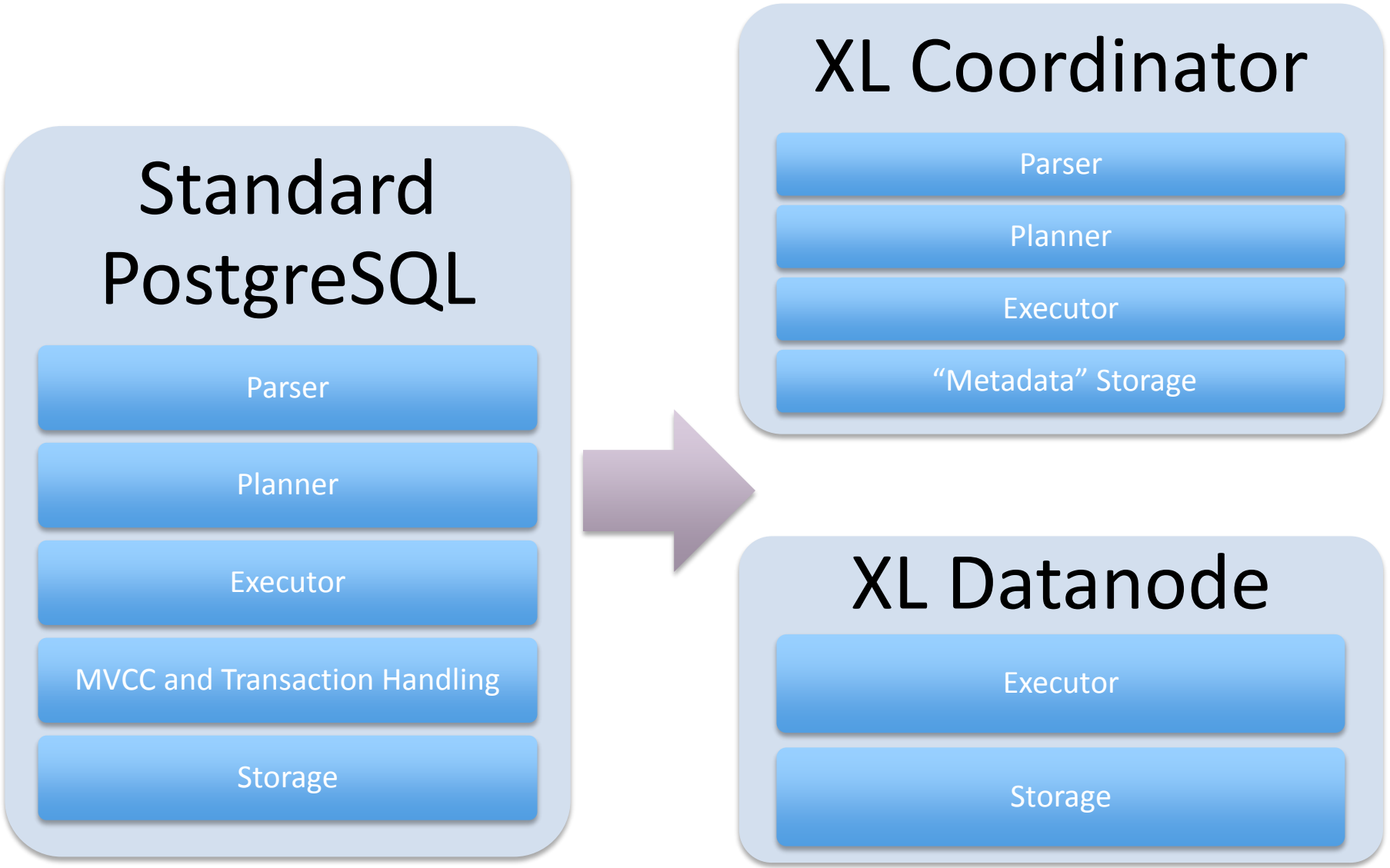
Postgres-XL Solution:

- Make XIDs and Snapshots cluster-wide


Load Balancer
Global Transaction Manager
Coordinators
Data Nodes

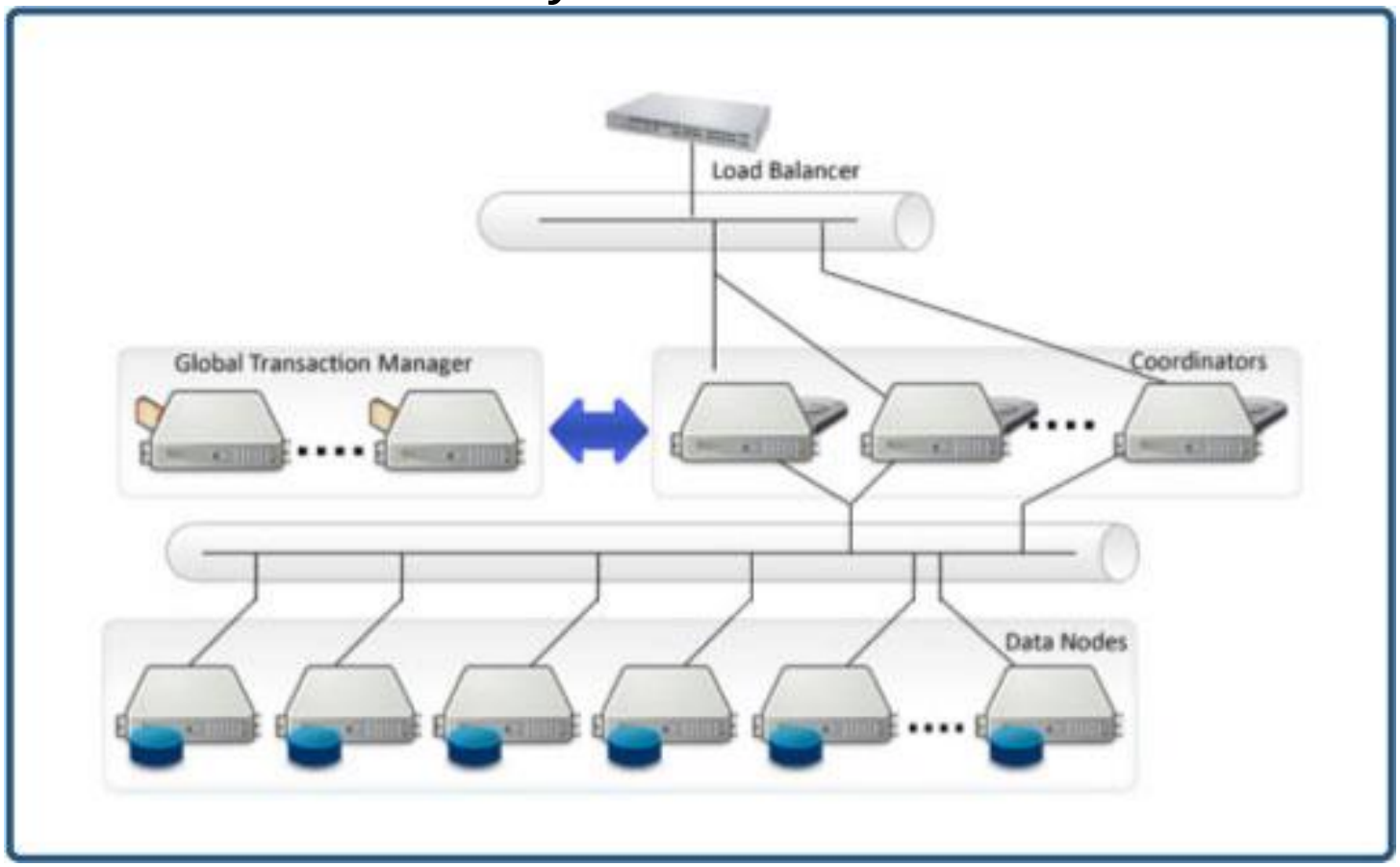

Standard
PostgreSQL
Parser
Planner
Executor
MVCC and Transaction Handling
Storage


Standard PostgreSQL
Parser
Planner
Executor
MVCC and Transaction Handling
Storage
GTM
MVCC and Transaction Handling
• Transaction ID (XID) Generation
• Snapshot Handling


Standard PostgreSQL
Parser
Planner
Executor
MVCC and Transaction Handling
Storage
XL Coordinator
Parser
Planner
Executor
“Metadata” Storage
XL Datanode
Executor
Storage

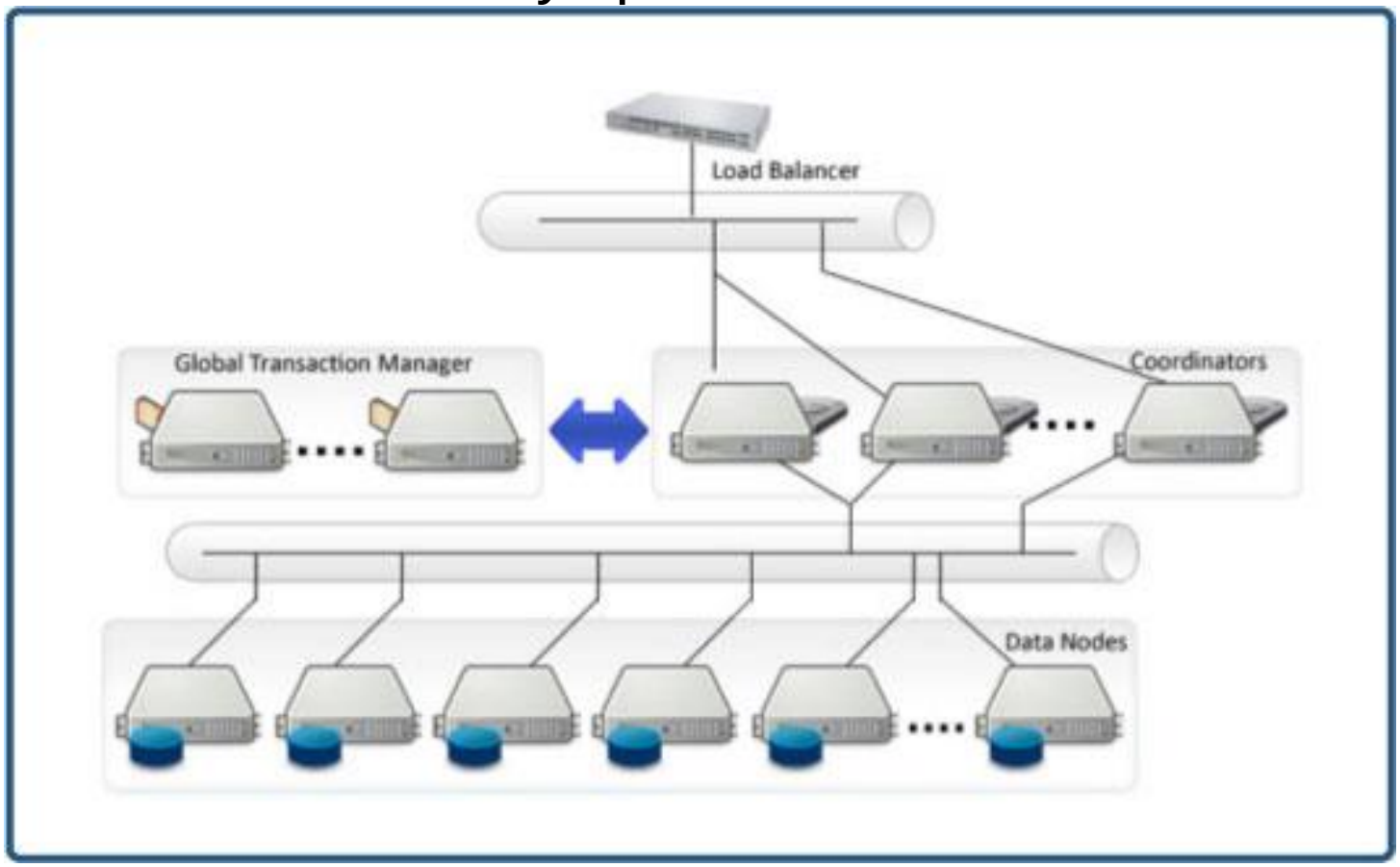
Postgres-XL Technology

- Users connect to any Coordinator



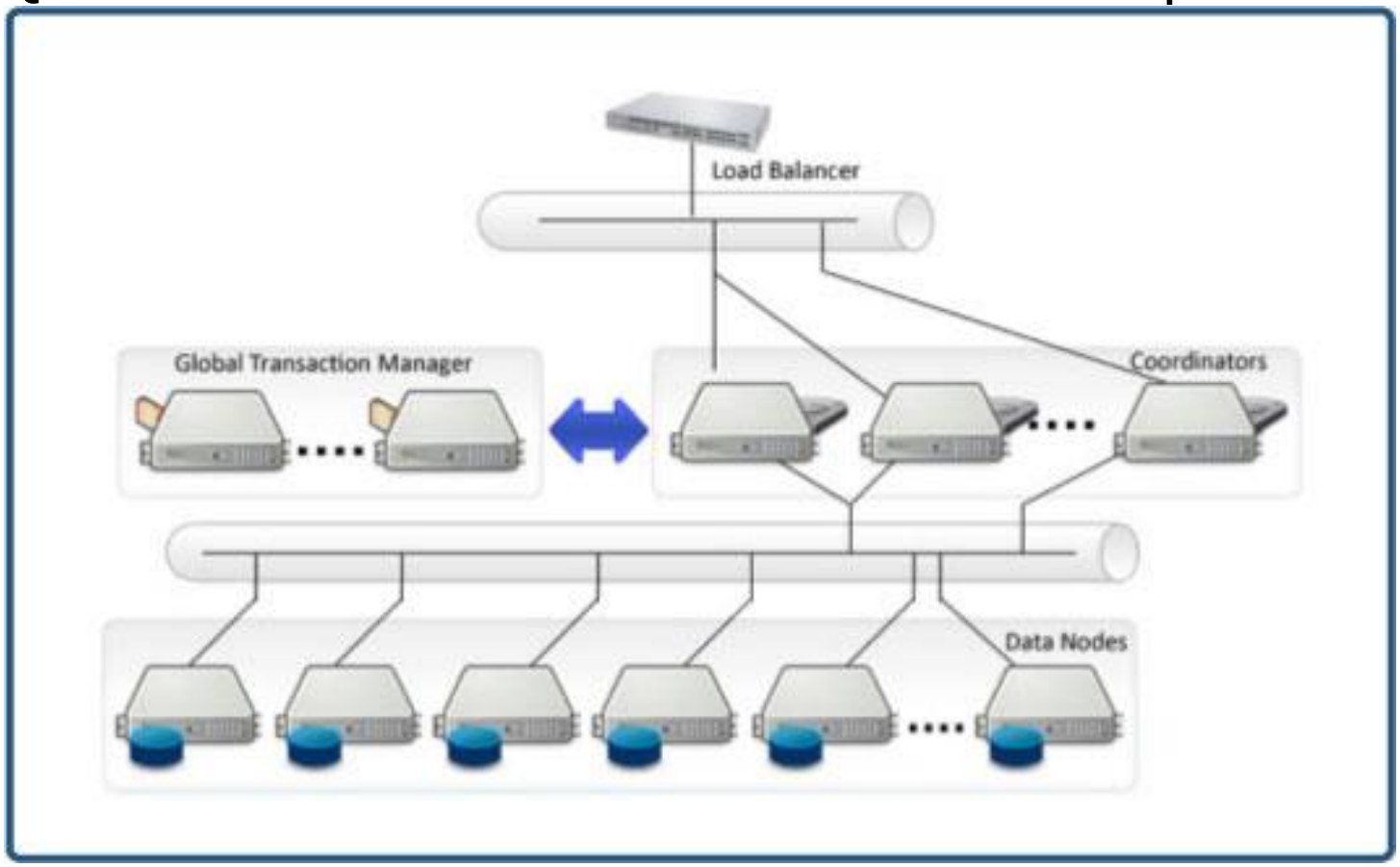
Postgres-XL Technology

- Data is automatically spread across cluster



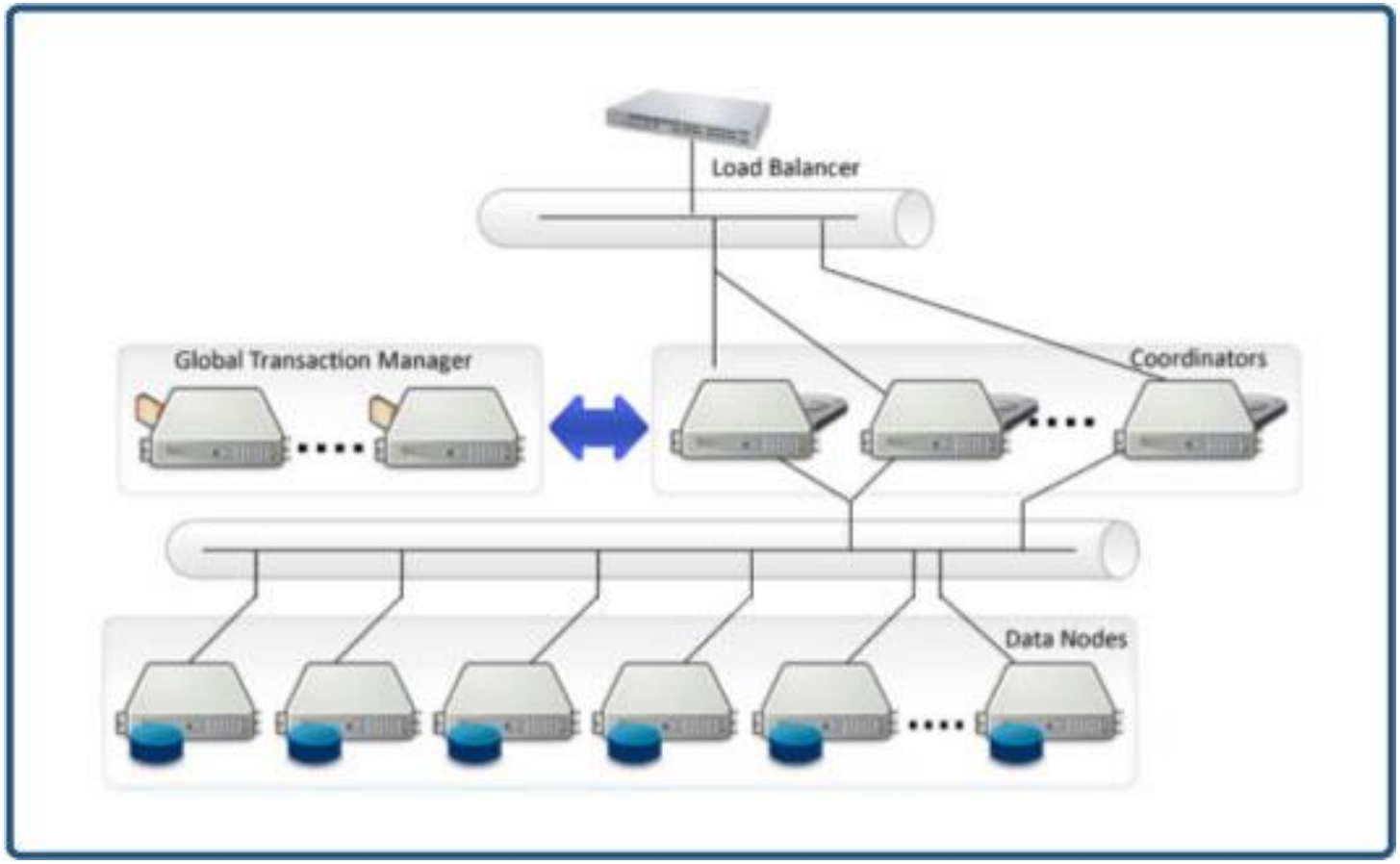
Postgres-XL Technology

- Queries return back data from all nodes in parallel



Postgres-XL Technology

- GTM maintains a consistent view of the data



Global Transaction Manager (GTM)

- Can this be a single point of failure?

Global Transaction Manager (GTM)

- Can this be a single point of failure?

Yes.

Global Transaction Manager (GTM)

- Can this be a single point of failure?

Yes.

Solution: GTM Standby

Global Transaction Manager (GTM)

- Can GTM be a bottleneck?

Global Transaction Manager (GTM)

- Can GTM be a bottleneck?

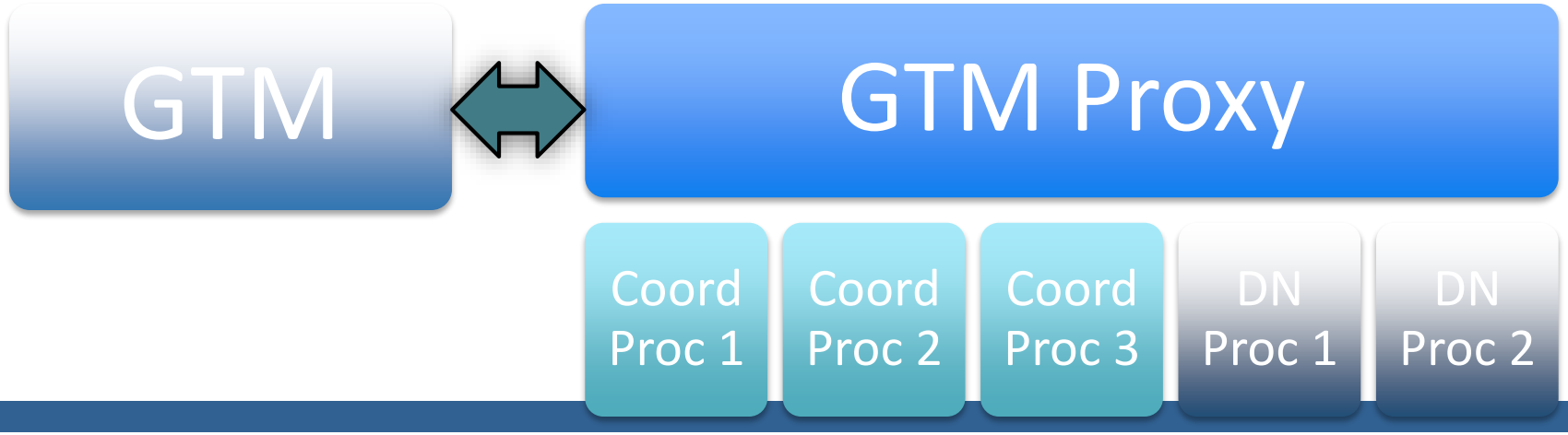
Yes.

Global Transaction Manager (GTM)

- Can GTM be a bottleneck?

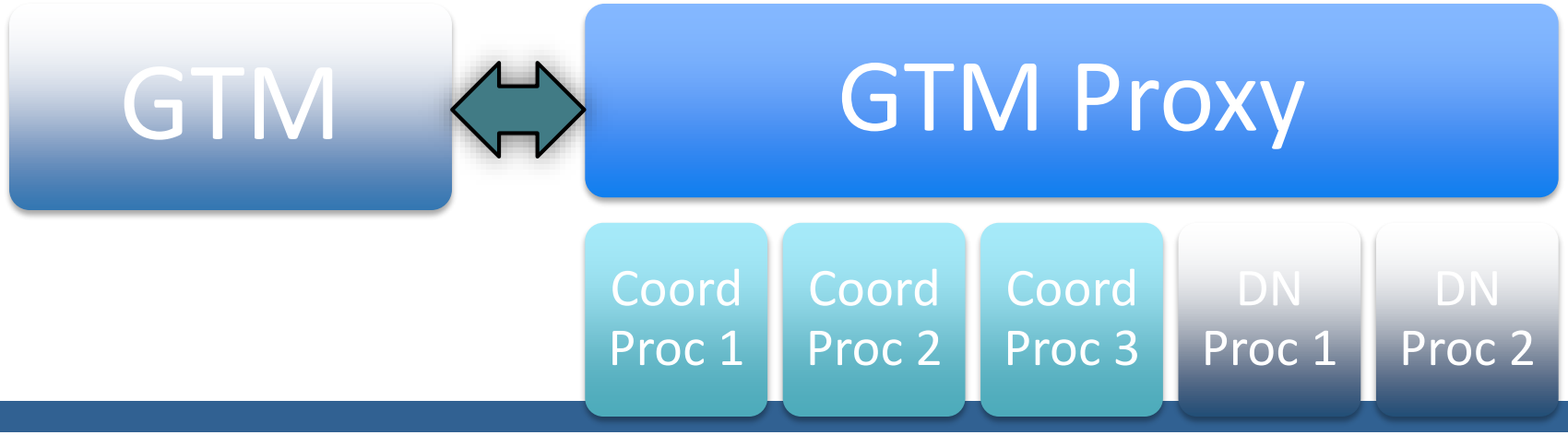
Yes.

Solution: GTM Proxy



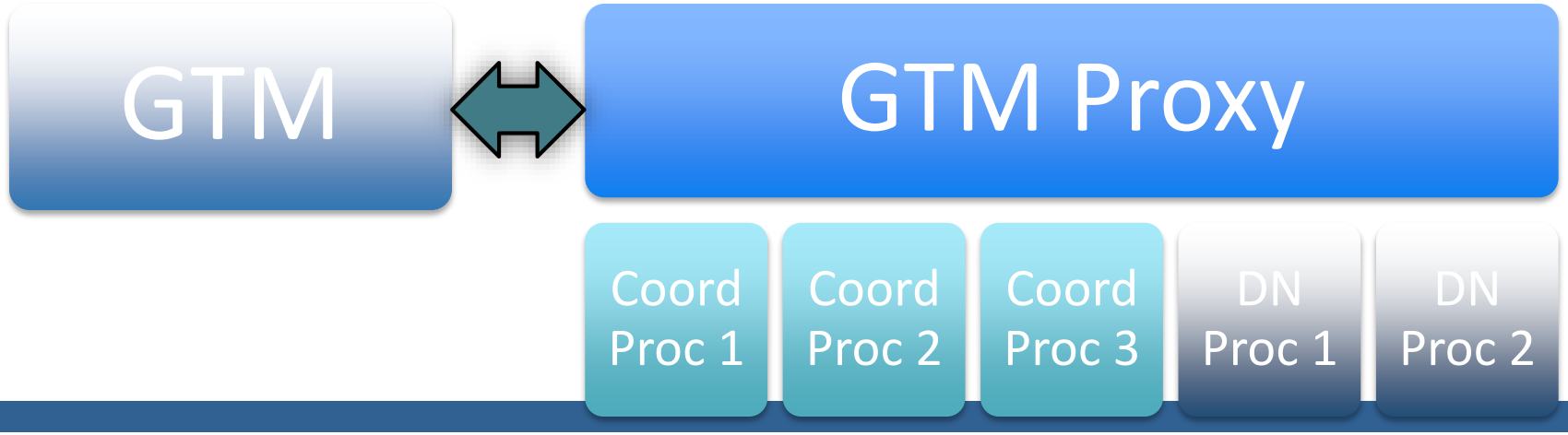
GTM Proxy

- Runs alongside Coordinator or Datanode
- Backends use it instead of GTM
- Groups requests together
- Obtain range of transaction ids (XIDs)
- Obtain snapshot



GTM Proxy

- Example: 10 process request a transaction id
- They each connect to local GTM Proxy
- GTM Proxy sends **one** request to GTM for 10 XIDs
- GTM locks procarray, allocates 10 XIDs
- GTM returns range
- GTM Proxy demuxes to processes

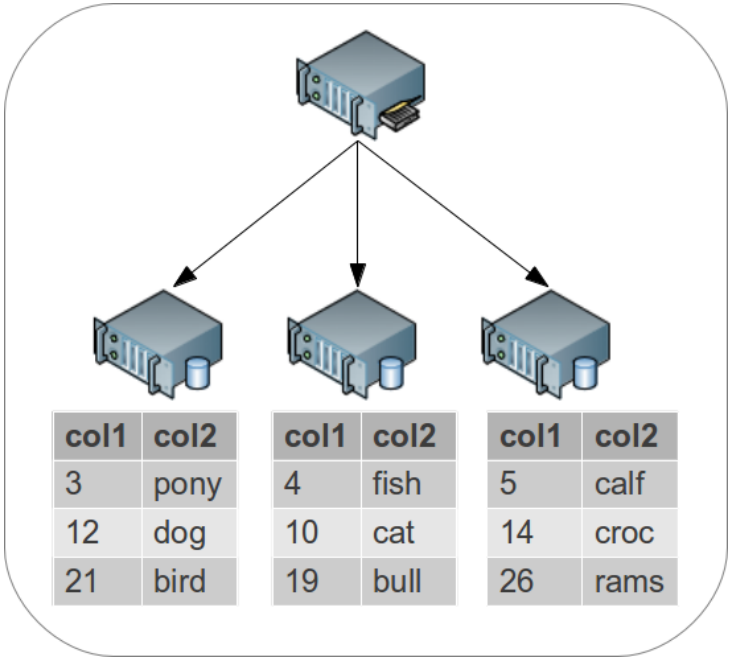


Data Distribution – Replicated Tables

- Good for read only and read mainly tables
- Sometimes good for dimension tables in data warehousing
- If coordinator and datanode are colocated, local read occurs
- Bad for write-heavy tables
- Each row is replicated to all data nodes
- Statement based replication
- “Primary” avoids deadlocks

Data Distribution – Distributed Tables

- Good for write-heavy tables
- Good for fact tables in data warehousing
- Each row is stored on one data node
- Available strategies
  - Hash
  - Round Robin
  - Modulo



Availability

- No Single Point of Failure
  - Global Transaction Manager Standby
  - Coordinators are load balanced
  - Streaming replication for data nodes
- But, currently manual…

DDL

Defining Tables

CREATE TABLE my_table (…)
DISTRIBUTE BY
HASH(col) | MODULO(col) | ROUNDROBIN | REPLICATION
[TO NODE (nodename[,nodename…])]

Defining Tables

```
CREATE TABLE state_code (
  state_code  CHAR(2),
  state_name VARCHAR,
  :
) DISTRIBUTE BY REPLICATION
```

An exact replica on each node

Defining Tables

```
CREATE TABLE invoice (
  invoice_id SERIAL,
  cust_id INT,
  :
) DISTRIBUTE BY HASH(invoice_id)
```

Distributed evenly amongst sharded nodes

Defining Tables

```
CREATE TABLE invoice (
  invoice_id SERIAL,
  cust_id INT ….
) DISTRIBUTE BY HASH(invoice_id);

CREATE TABLE lineitem (
  lineitem_id SERIAL,
  invoice_id INT …
) DISTRIBUTE BY HASH(invoice_id);
```

Joins on invoice_id can be pushed down to the datanodes

test2=# create table t1 (col1 int, col2 int) **distribute by hash(col1);**

test2=# create table t2 (col3 int, col4 int) **distribute by hash(col3);**

explain select * from t1 inner join t2 o

Join push-down. Looks much like regular PostgreSQL

Remote Subquery Scan on all (**datanode_1,datanode_2**)

-> Hash Join

Hash Cond:

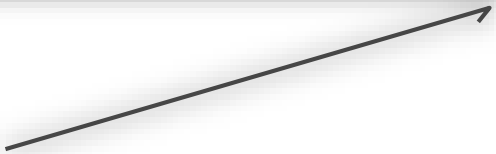
-> Seq Scan on t1

-> Hash

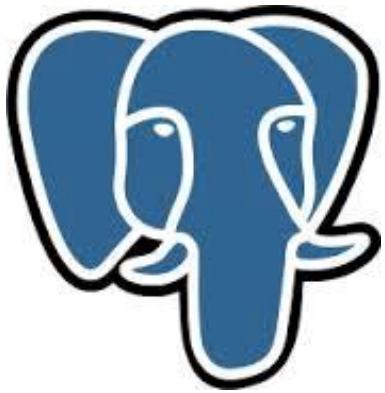
-> Seq Scan on t2

```
test2=# explain select * from  t1 inner jo
col3;

Remote Subquery Scan on all (datano
  ->  Hash Join
        Hash Cond:
        ->  Remote Subquery Scan on all
(datanode_1,datanode_2)
              Distribute results by H: col2
              ->  Seq Scan on t1
        ->  Hash
              ->  Seq Scan on t2
```

Will read t1.col2 once and put in shared queue for consumption for other datanodes for joining. Datanode-datanode communication and parallelism



PostgreSQL Differences



pg_catalog

- pgxc_node
  - Coordinator and Datanode definition
  - Currently not GTM
- pgxc_class
  - Table replication or distribution info

Additional Commands

- PAUSE CLUSTER / UNPAUSE CLUSTER
  - Wait for current transactions to complete
  - Prevent new commands until UNPAUSE
- EXECUTE DIRECT

  ON (nodename) ‘command’
- CLEAN CONNECTION
  - Cleans connection pool
- CREATE NODE / ALTER NODE

Noteworthy

- SELECT pgxc_pool_reload()
- pgxc_clean for 2PC

Configuration:

Use pgxc_ctl!

(please get from git HEAD for latest bug fixes)

Otherwise, PostgreSQL-like steps apply:

initgtm

initdb

postgresql.conf

- Very similar to regular PostgreSQL
- But there are extra parameters
  - max_pool_size
  - min_pool_size
  - pool_maintenance _timeout
  - remote_query_cost
  - network_byte_cost
  - sequence_range
  - pooler_port
  - gtm_host, gtm_port
  - shared_queues
  - shared_queue_size

Thanks!

Q & A