

Postgres-XC: Write-Scalable PostgreSQL Cluster

NTT Open Source Software Center
EnterpriseDB Corp.

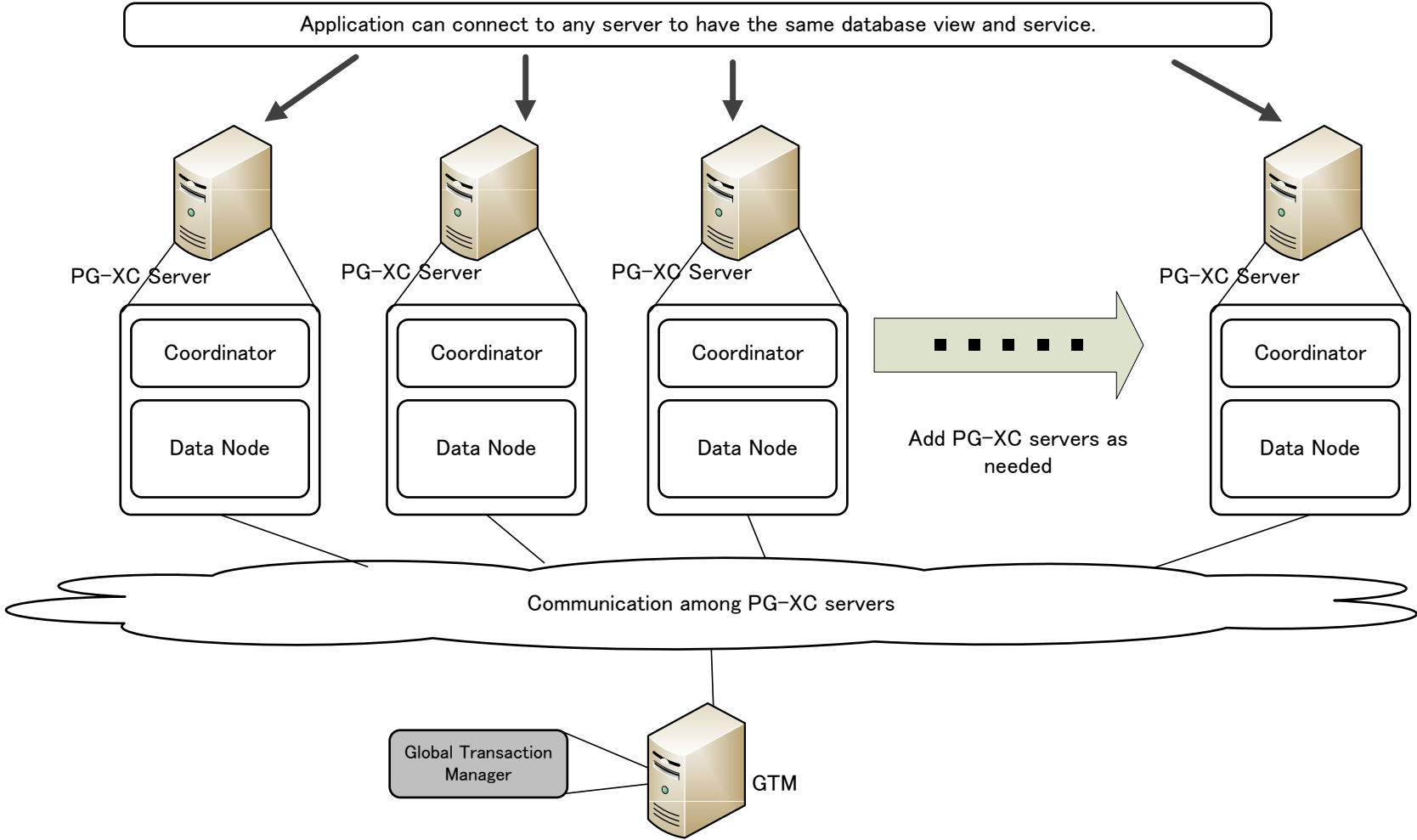
What is Postgres-XC (or PG-XC)?

- Write-scalable PostgreSQL cluster
 - More than 3.4 performance scalability with five servers, compared with pure PostgreSQL (DBT-1)
- Synchronous multi-master configuration
 - Any update to any master is visible from other masters immediately.
- Table location transparent
 - Can continue to use the same applications.
 - No change in transaction handling.
- Based upon PostgreSQL
- Same API to Apps. as PostgreSQL

Why write-scalability?

- Many application could be write-traffic bottleneck such as –
 - Access log in BLOG/SNS
 - Mission critical systems like internet shopping site, telephone customer billing, call information and securities trade
- Now application has to deal with such write-bottleneck using multi-database.
 - Not distribution-transparent.
- As applications grow
 - It is desirable to make database distribution transparent for write operations too.

Postgres-XC Architecture Outline



Postgres-XC Architecture

- Shared-nothing architecture
 - No shared disk
 - No shared memory
 - Only communication infrastructure
- Three Components
 - GTM (Global Transaction Manager)
 - Provide global transaction information to each transaction
 - Transaction ID
 - Snapshot
 - Provide other global data to statements
 - Sequence
 - Time/Sysdate (under plan)
 - Coordinator
 - Parse statements and determine location of involved data
 - Transfer statements for each data node (if needed)
 - Application I/F
 - Data Node
 - Store actual data
 - Execute statements from Coordinators

Postgres-XC also has Pooler to reuse coordinator and data node connections.

What Applications?

- Short transaction applications (DBT-1/2 etc.)
 - Transactions can be executed in parallel in multiple data nodes.
- Complicated data warehouse (DBT-3 etc.)
 - Statement can be divided into several pieces which can be executed in parallel in multiple data nodes.
 - (Statement handling not available yet.)

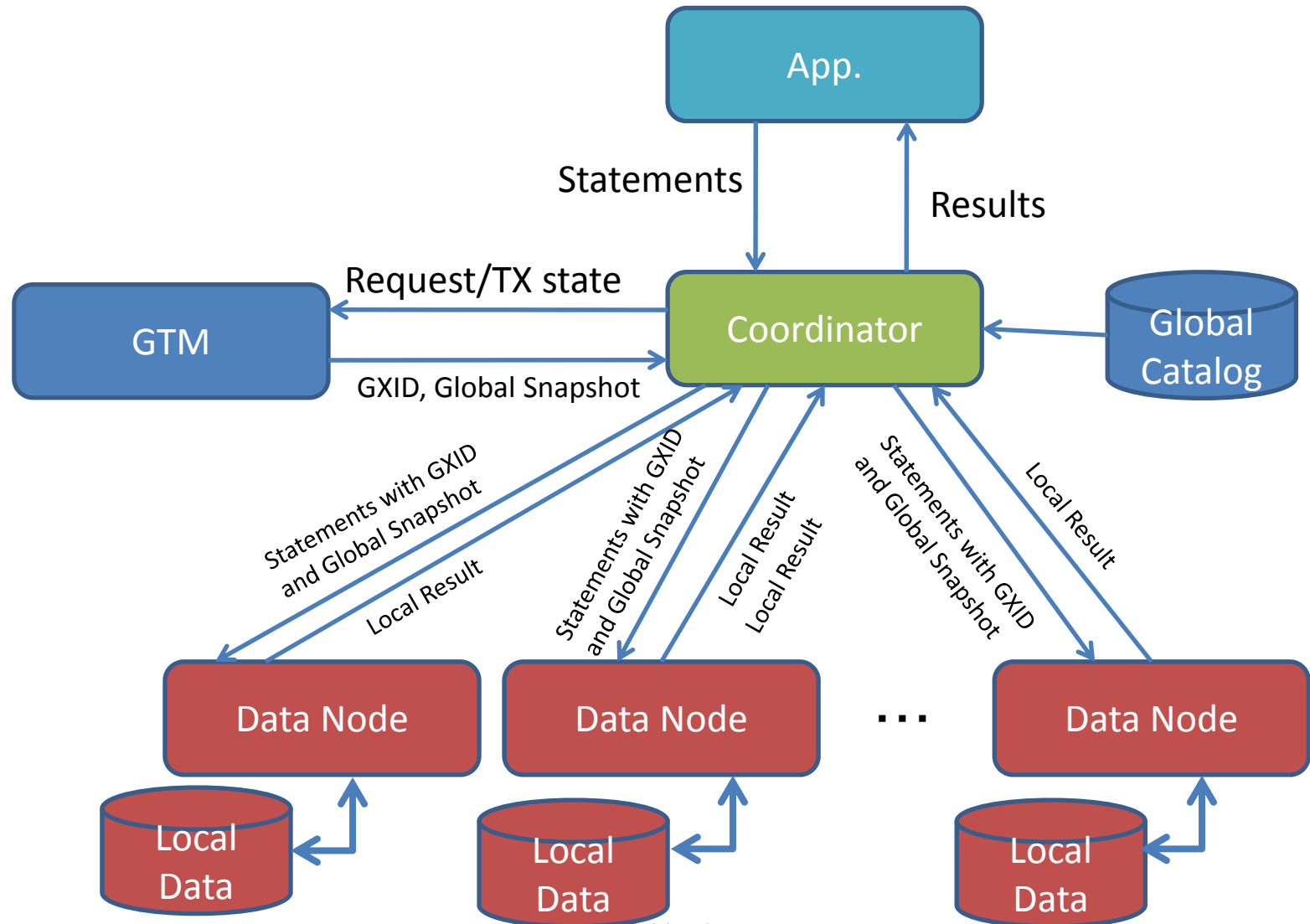
How to distribute tables?

- Tables can be partitioned or replicated over PG-XC servers according to application needs.
 - Can select partitioning key.
 - Rows will be partitioned according to the key value.
 - Hash
 - Range (future)
 - Others (future)
 - Transaction tables may be partitioned so that each transaction can be executed in limited number of data nodes.
 - Master tables may be replicated so that each transaction can read row values locally.
 - Table partitioning/replication is defined in the global catalog maintained by the coordinator.

GTM: A Key Component

- Extracted essential of transaction management feature of PostgreSQL
 - Unique Transaction ID (GXID, Global Transaction ID) assignment,
 - Gather transaction status from all the coordinators and maintain snapshot data,
 - Provide snapshot data to each transaction/statement.
- Extract global value providing feature such as
 - Sequence
 - Time/sysdate

Components involved in a transaction



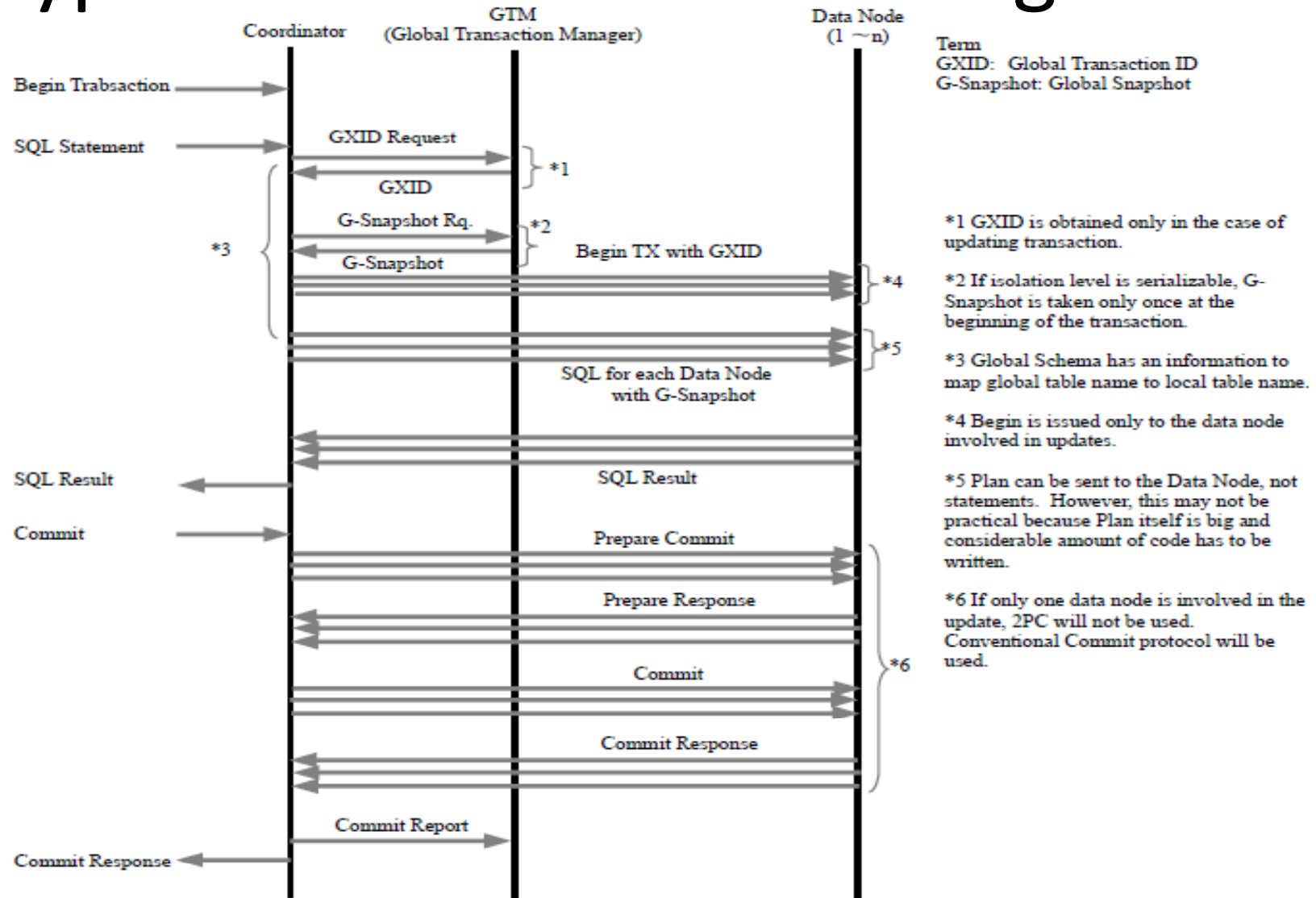
GXID and Global Snapshot

- GXID
 - Unique Transaction ID in the system
- Global Snapshot
 - Includes snapshot information of transactions in other coordinators.



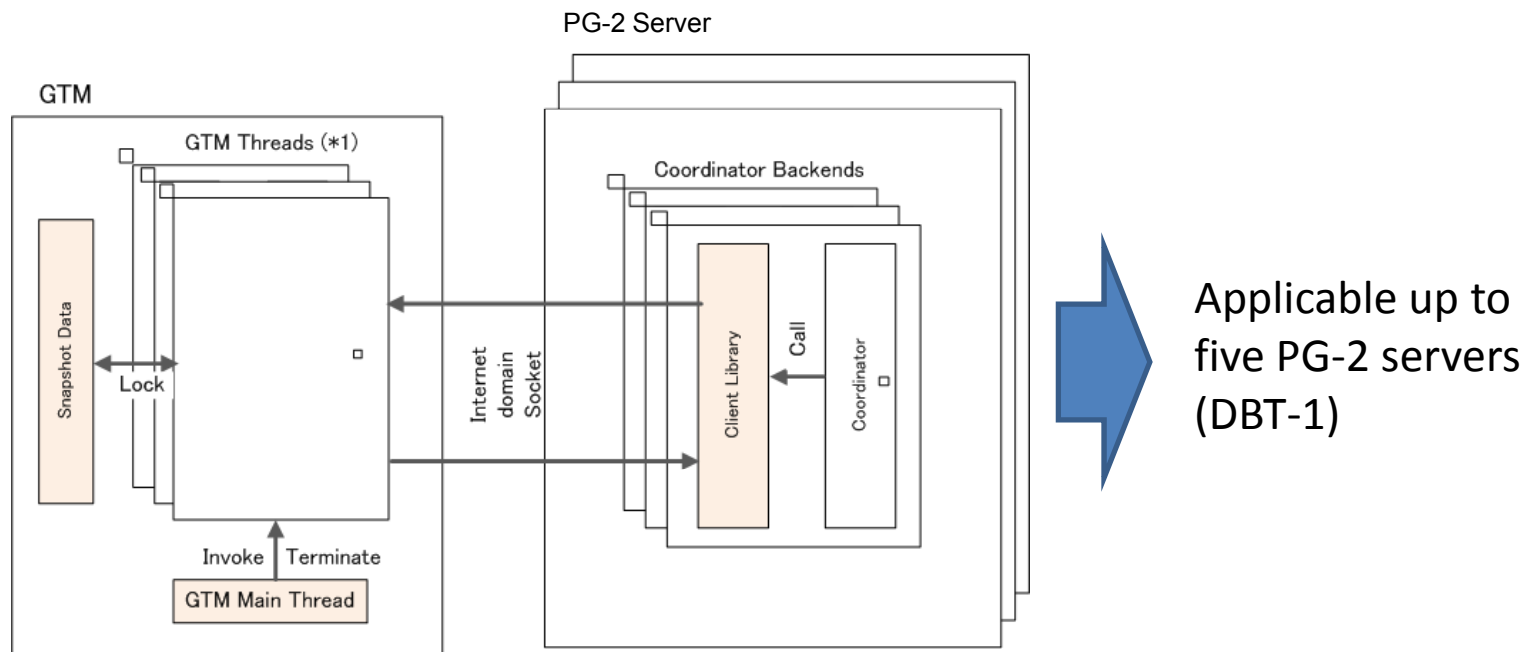
- Data node can handle transactions from different coordinators without consistency problem.
- Visibility is maintained as standalone PostgreSQL.

Typical transaction handling flow



Could GTM be a bottleneck?

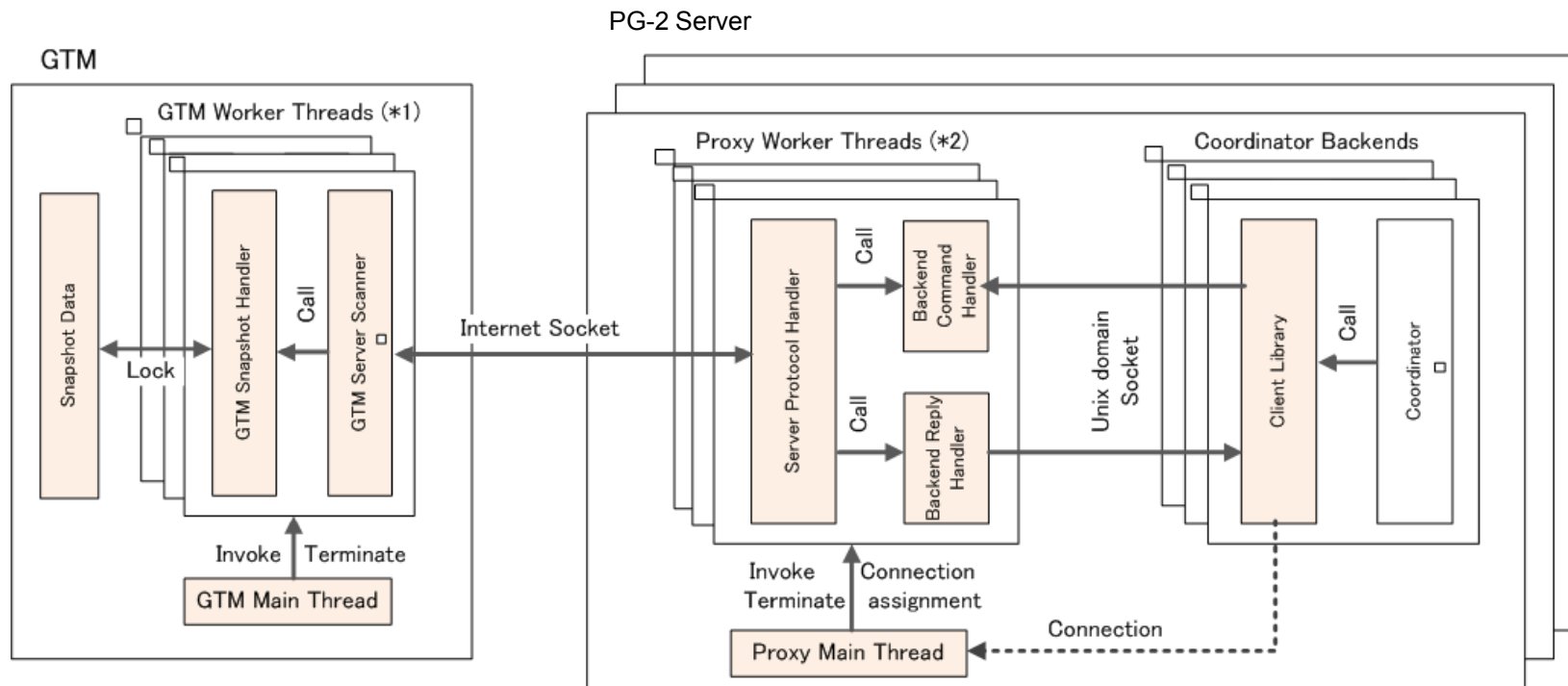
- Depending on implementation
 - Current Implementation



- Large snapshot size and number
- Too many interaction between GTM and Coordinators

Could GTM be a bottleneck (cont.)?

- Proxy Implementation

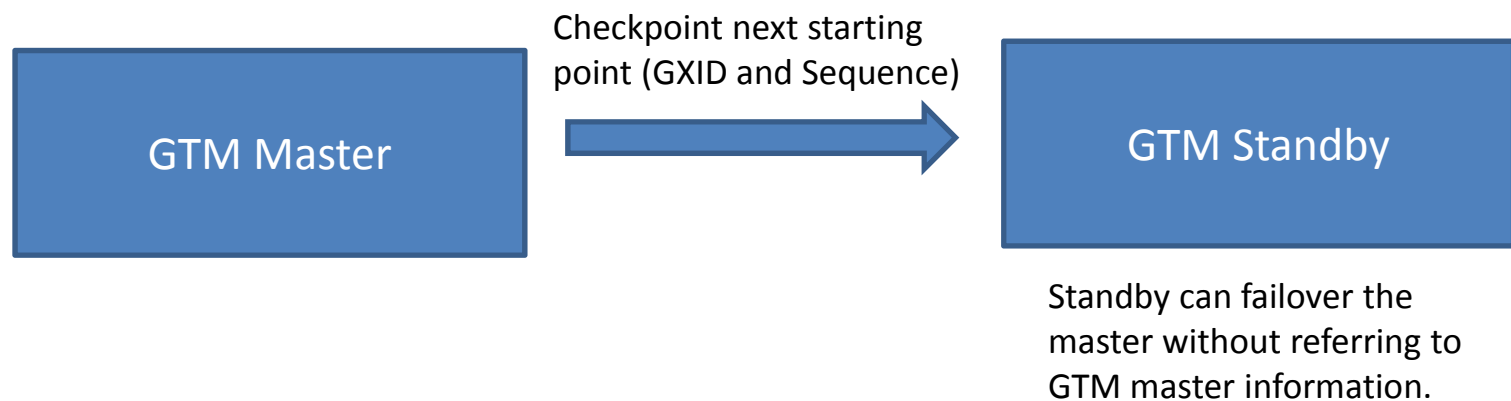


*1 GTM Server Worker Thread is created when new connection from the proxy is accepted.
*2 Number of Proxy Worker Thread is specified when Proxy Main Thread is invoked.

- Very good potential
 - Request/Response grouping
 - Single representative snapshot applied to multiple transactions
- Maybe applicable for more than ten PG-2 servers

Could GTM be a SPOF?

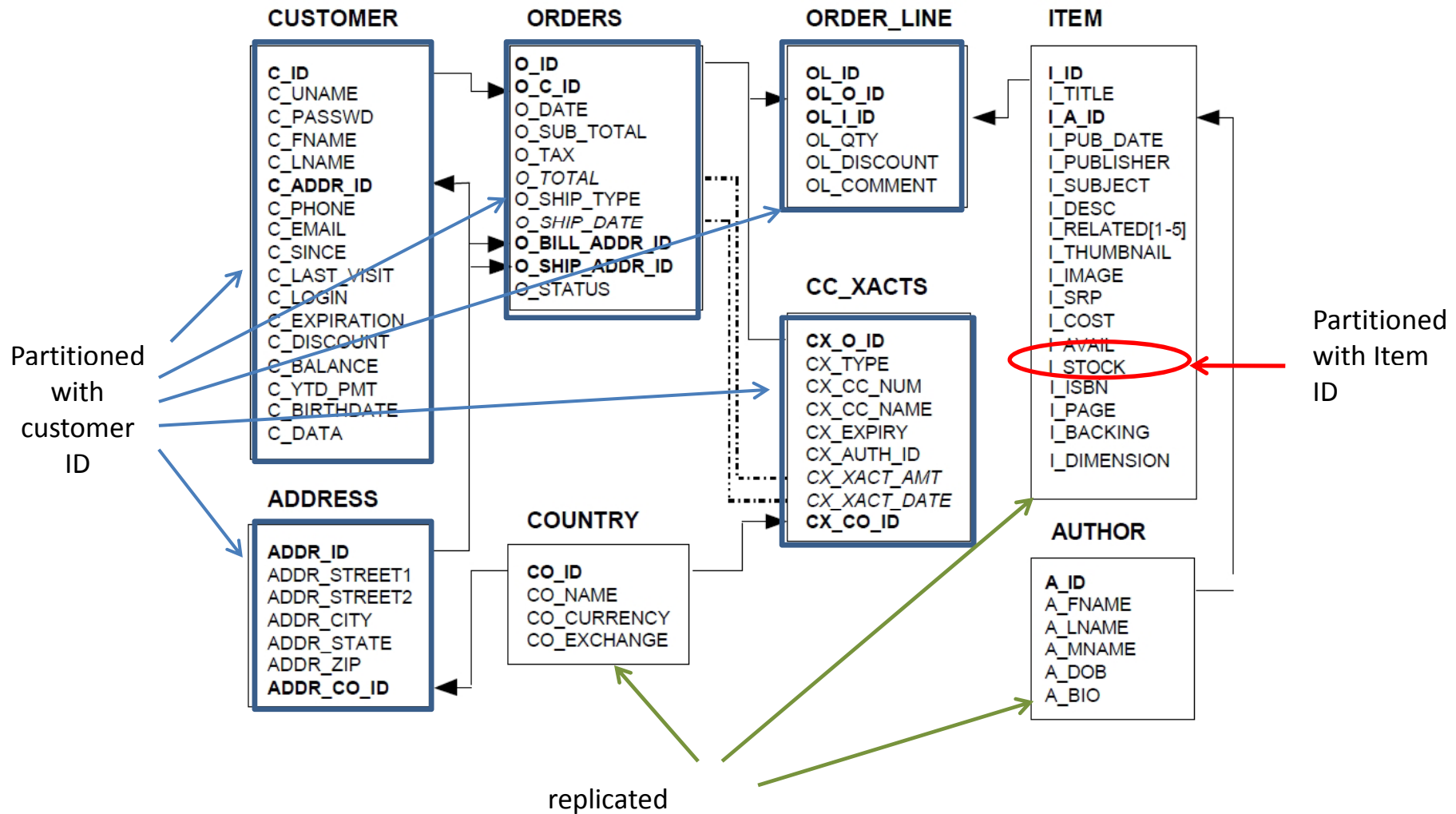
- Simple to implement GTM standby



DBT-1 Performance Benchmark

- DBT-1 schema change manually for partitioning
 - DDL not yet unavailable
 - Utilize key dependence
 - Added joins to WHERE clauses if needed
 - Could be handled automatically when DDL is supported
- Three replicated tables
- Seven partitioned tables
 - Three partitioning keys
- Item table is divided into item and inventory
 - As found in new TPC-W spec.

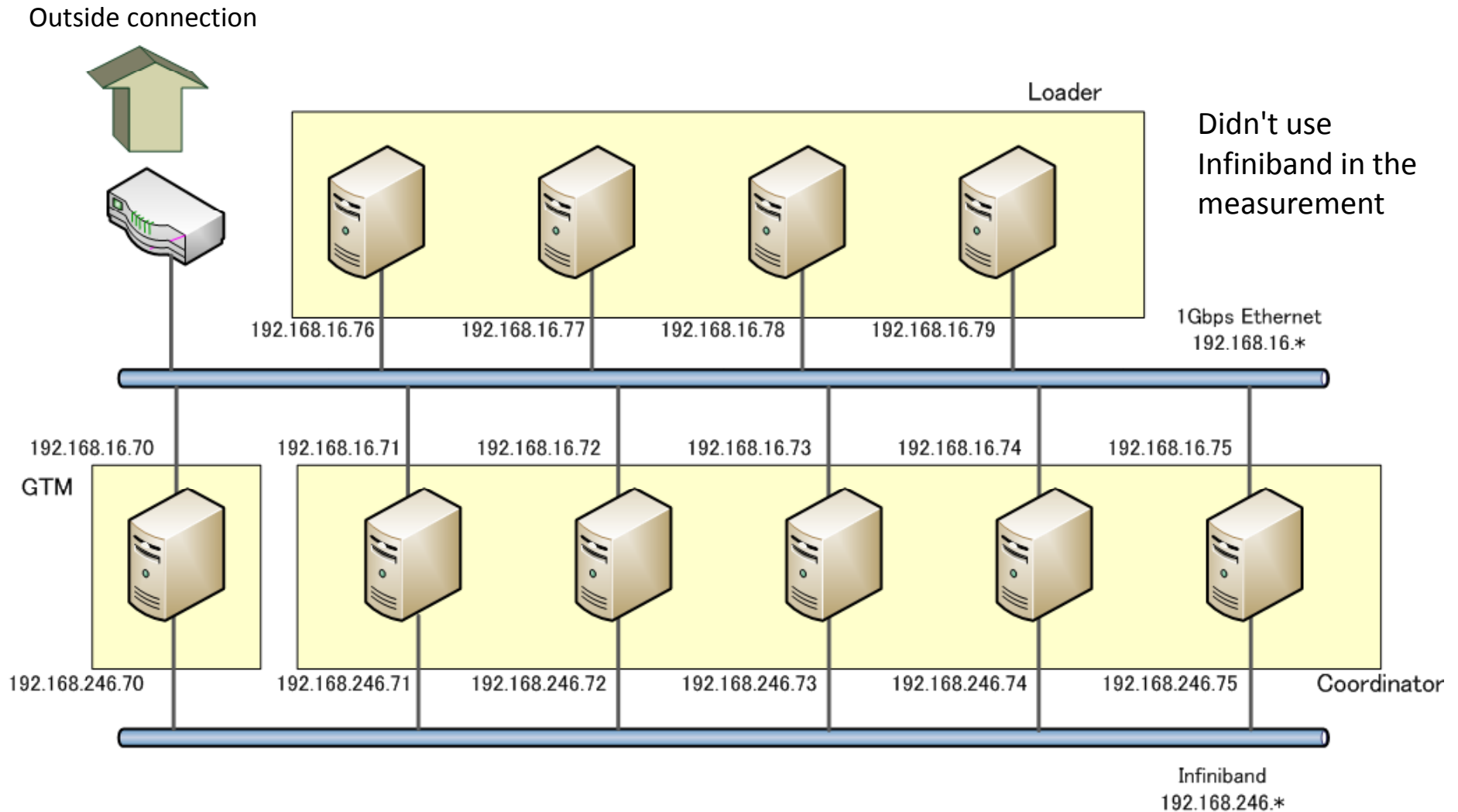
DBT-1 Performance Benchmark (cont.1)



DBT-1 Performance Benchmark (cont.2)

- Shopping cart and Shopping cart line
 - Partitioned using shopping cart ID

Performance Measurement Environment



Throughput

Configuration	Performance	Relative to PostgreSQL	Relative to single node PG-2
Pure PostgreSQL	2500 TPS	1	1.32 or worse*
Single Node PG-2	1740 TPS	0.76 or better*	1
Five Node PG-2	8140 TPS	3.4 or better*	4.4 or better*

- Very good performance
- Scale factor is excellent
 - May scale up to ten nodes.
- No significant performance drop in single node PG-2.
- Does not scale linearly from single node to five nodes
 - Additional communication among PG-2 servers
 - Additional overhead by 2PC (maybe very small)

*Above score is the worst one, when original PostgreSQL setting consumes almost 100% CPU. If original setting consumes less, scalability is better.

Current Implementation

- Minimum feature to run DBT-1
 - No backup/recovery
 - Minimum error handling
 - Use timeout to detect cross-node deadlocks
 - Minimum SQL feature
 - No DDL
 - Global catalog setup manually
 - Manual table creation in each node
 - Hash partitioning only
 - Range partitioning not available yet
 - No cross-node join (not necessary in DBT-1)
 - No aggregate functions
 - No "copy"
 - Partitioning keys cannot be updated
 - Need to relocate tuples.
 - No consistent update of replicated tables
 - DBT-1 does not update replicated tables
 - Pgpool-II methodology can be applied.
 - 2PC improvement
 - Saved writes to state files
 - Writes to state files occur if a transaction is left prepared and not committed or aborted at checkpoints.

Future issues

- Stabilize the code
 - Continue to run with full load for days/weeks
- Coordinator enhancement
- Open the code
 - Can GTM be used in other projects to harmonize multi-master synchronously?
- Integration with future PostgreSQL releases
 - APIs?
 - Hooks?
 - Can reuse PostgreSQL binaries?