EnterpriseDB®
The Enterprise PostgreSQL Company

Scaling out by distributing and replicating data in Postgres-XC

Ashutosh Bapat

@Postgres Open 2012

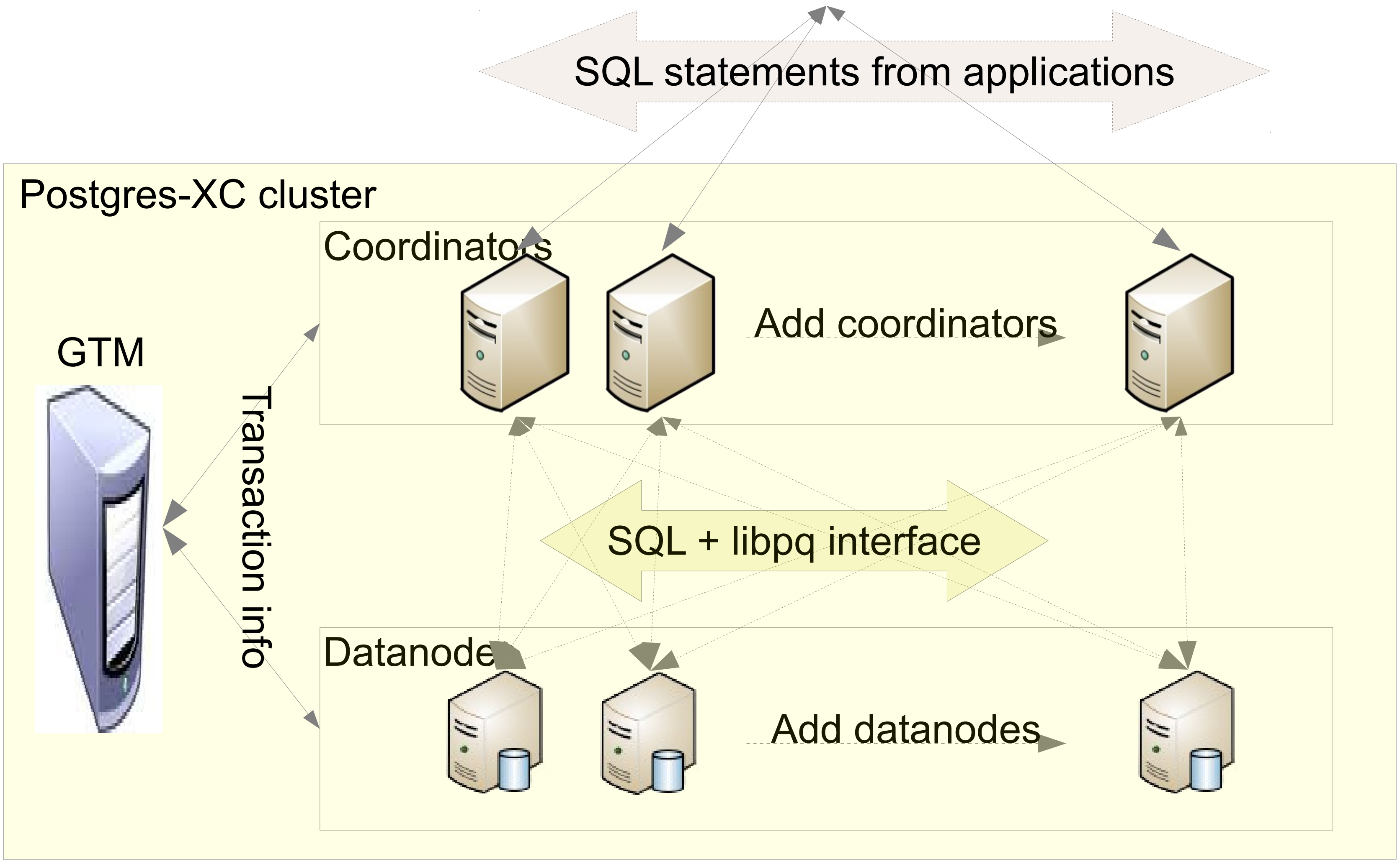
Agenda

- What is Postgres-XC
- Postgres-XC architecture over-view
- Data distribution in XC
- Effect of data distribution on performance
- Example DBT-1 schema

What is Postgres-XC

- Shared Nothing Cluster
  - Multiple collaborating PostgreSQL-like servers
  - No resources shared
  - Scaling by adding commodity hardware
- Write-scalable
  - Write/Read scalable by adding nodes
  - Multiple nodes where writes can be issued
- Syncronous
  - Writes to one node are reflected on all the nodes
- Transparent
  - Applications need not care about the data distribution

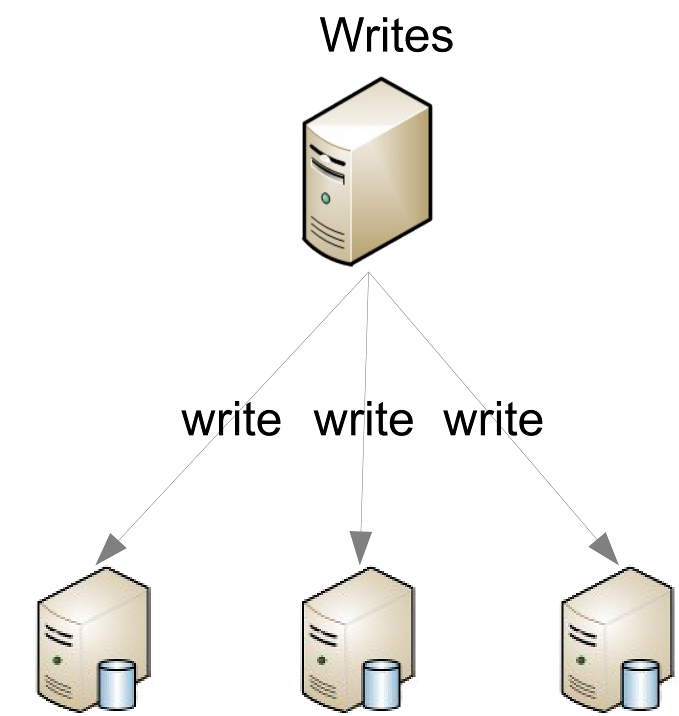
Postgres-XC architecture



Distribution strategies

- Replicated tables
  - Each row of the table is stored on all the datanodes where the table is replicated
- Distributed tables
  - Each row exists only on a single datanode
  - Distribution strategies
    - HASH
    - MODULO
    - ROUNDROBIN
    - User defined functions (TBD)

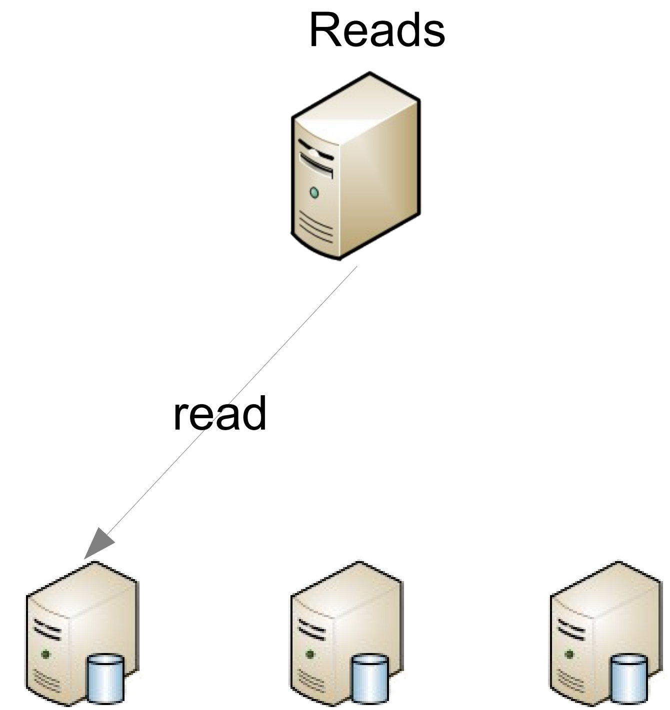
Replicated Table



val	val2
1	2
2	10
3	4

val	val2
1	2
2	10
3	4

val	val2
1	2
2	10
3	4



val	val2
1	2
2	10
3	4

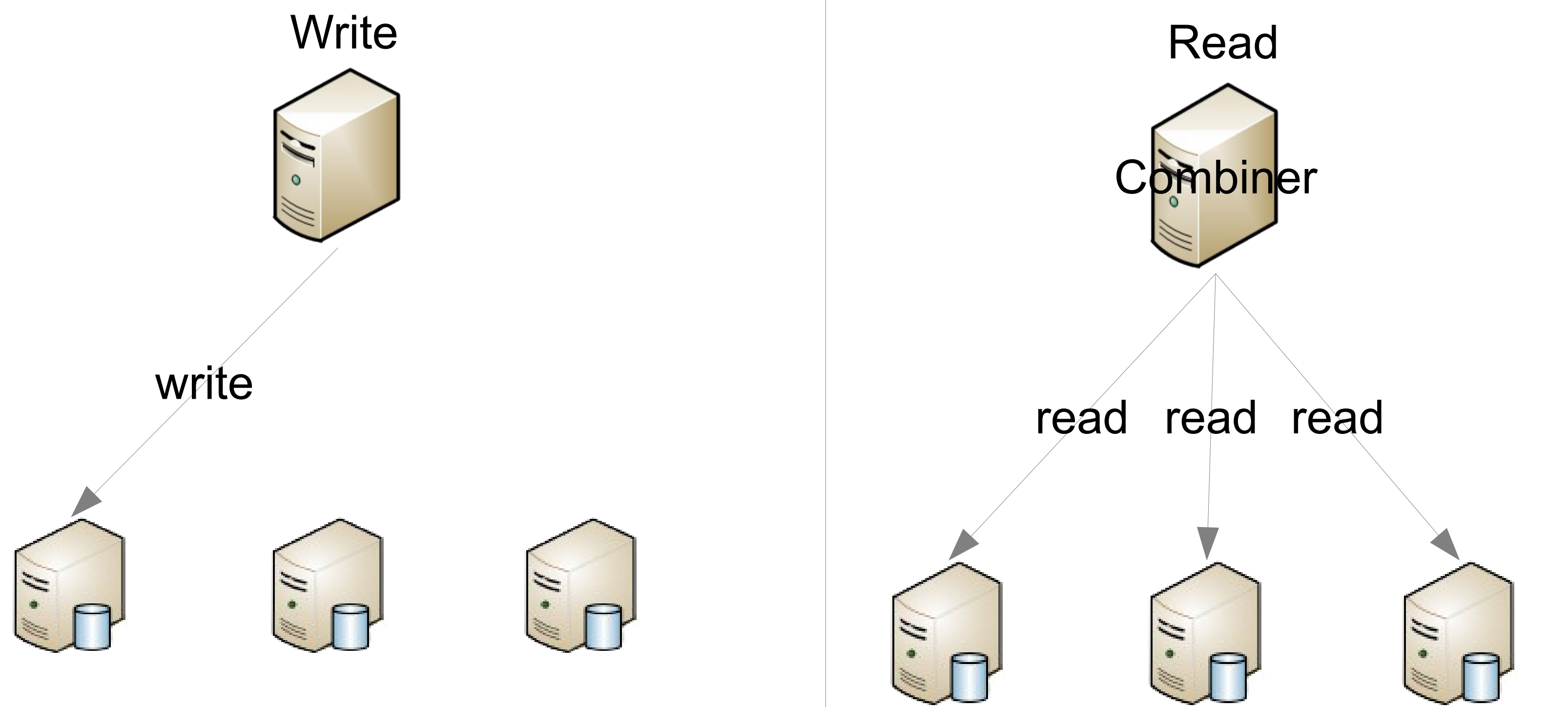
val	val2
1	2
2	10
3	4

val	val2
1	2
2	10
3	4

Replicated Tables

- Statement level replication
- Each write needs to be replicated
  - writes are costly
- Read can happen on any node (where table is replicated)
  - reads from different coordinators can be routed to different nodes
- Useful for relatively static tables, with high read load

Distributed Tables



val	val2
1	2
2	10
3	4

val	val2
11	21
21	101
31	41

val	val2
10	20
20	100
30	40

val	val2
1	2
2	10
3	4

val	val2
11	21
21	101
31	41

val	val2
10	20
20	100
30	40

Distributed Tables

- Write to a single row is applied only on the node where the row resides
  - Multiple rows can be written in parallel
- Scanning rows spanning across the nodes (e.g. table scans) can hamper performance
- Point reads and writes based on the distribution column value show good performance
  - Datanode where the operation happens can be identified by the distribution column value

Distributed query processing in Postgres-XC

Distributed query processing in Postgres-XC

- Coordinator
  - Accepts queries and plans them
  - Finds the right data-nodes from where to fetch the data
  - Frames queries to be sent to these data-nodes
  - Gathers data from data-nodes
  - Processes it to get the desired result
- Datanode
  - Executes queries from coordinator like PostgreSQL
  - Has same capabilities as PostgreSQL

Query processing balance

- Coordinator tries to delegate maximum query processing to data-nodes
  - Indexes are located on datanodes
  - Materialization of huge results is avoided in case of sorting, aggregation, grouping, JOINs etc.
  - Coordinator is freed to handle large number of connections
- Distributing data wisely helps coordinator to delegate maximum query processing and improve performance
- Delegation is often termed as shipping

SQL prompt

Deciding the right distribution strategy

Read-write load on tables

- High point reads (based on distribution column)
  - Distributed or replicated
- High read activities but no frequent writes
  - Better be replicated
- High point writes
  - Better be distributed
- High insert-load, but no frequent update/delete/read
  - Better be round-robin

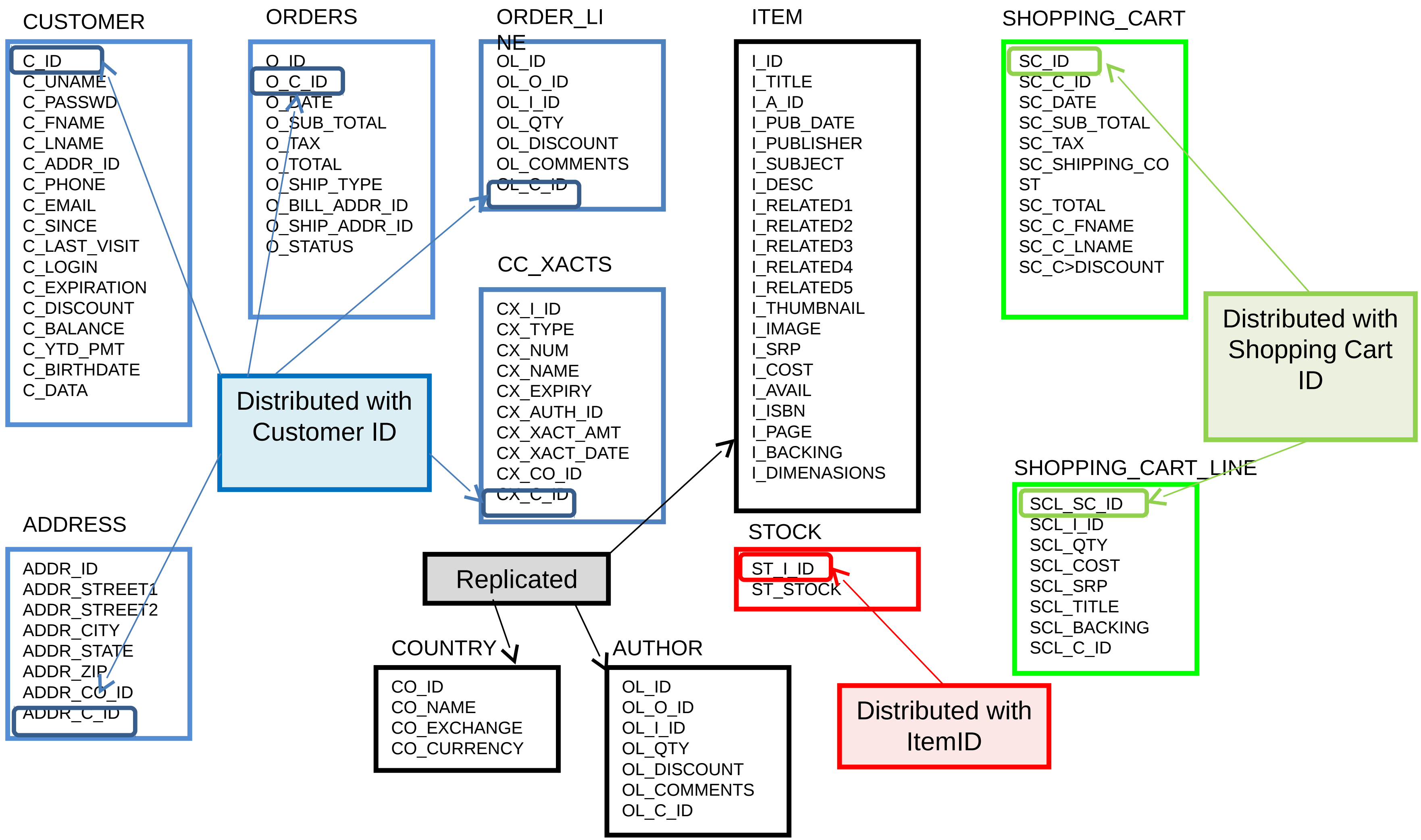
Query analysis (Frequently occuring queries)

- Find the relations/columns participating in equi-Join conditions, WHERE clause etc.
  - Distribute on those columns
- Find columns participating in GROUP BY, DISTINCT clauses
  - Distribute on those columns
- Find columns/tables which are part of primary key and foreign key constraints
  - Global constraints are not yet supported in XC
  - Distribute on those columns

Thumb rules

- Infrequently written tables participating in JOINs with many other tables (Dimension tables)
  - Replicated table
- Frequently written tables participating in JOINs with replicated tables
  - Distributed table
- Frequently written tables participating in JOINs with each other, with equi-JOINing columns of same data type
  - Distribute both of them by the columns participating in JOIN on same nodes
- Referenced tables
  - Better be replicated

DBT-1 schema


CUSTOMER
C_ID
C_UNAME
C_PASSWD
C_FNAME
C_LNAME
C_ADDR_ID
C_PHONE
C_EMAIL
C_SINCE
C_LAST_VISIT
C_LOGIN
C_EXPIRATION
C_DISCOUNT
C_BALANCE
C_YTD_PMT
C_BIRTHDATE
C_DATA
ORDERS
O_ID
O_C_ID
O_DATE
O_SUB_TOTAL
O_TAX
O_TOTAL
O_SHIP_TYPE
O_BILL_ADDR_ID
O_SHIP_ADDR_ID
O_STATUS
ORDER_LINE
OL_ID
OL_O_ID
OL_I_ID
OL_QTY
OL_DISCOUNT
OL_COMMENTS
OL_C_ID
ITEM
I_ID
I_TITLE
I_A_ID
I_PUB_DATE
I_PUBLISHER
I_SUBJECT
I_DESC
I_RELATED1
I_RELATED2
I_RELATED3
I_RELATED4
I_RELATED5
I_THUMBNAIL
I_IMAGE
I_SRP
I_COST
I_AVAIL
I_ISBN
I_PAGE
I_BACKING
I_DIMENASIONS
SHOPPING_CART
SC_ID
SC_C_ID
SC_DATE
SC_SUB_TOTAL
SC_TAX
SC_SHIPPING_COST
SC_TOTAL
SC_C_FNAME
SC_C_LNAME
SC_C>DISCOUNT
Distributed with Shopping Cart ID
CC_XACTS
CX_I_ID
CX_TYPE
CX_NUM
CX_NAME
CX_EXPIRY
CX_AUTH_ID
CX_XACT_AMT
CX_XACT_DATE
CX_CO_ID
CX_C_ID
Distributed with Customer ID
ADDRESS
ADDR_ID
ADDR_STREET1
ADDR_STREET2
ADDR_CITY
ADDR_STATE
ADDR_ZIP
ADDR_CO_ID
ADDR_C_ID
Replicated
STOCK
ST_I_ID
ST_STOCK
SHOPPING_CART_LINE
SCL_SC_ID
SCL_I_ID
SCL_QTY
SCL_COST
SCL_SRP
SCL_TITLE
SCL_BACKING
SCL_C_ID
COUNTRY
CO_ID
CO_NAME
CO_EXCHANGE
CO_CURRENCY
AUTHOR
OL_ID
OL_O_ID
OL_I_ID
OL_QTY
OL_DISCOUNT
OL_COMMENTS
OL_C_ID
Distributed with ItemID

Example DBT-1 (1)

- author, item
  - Less frequently written
  - Frequently read from
  - Author and item are frequently JOINed
    - Dimension tables
  - Hence replicated on all nodes

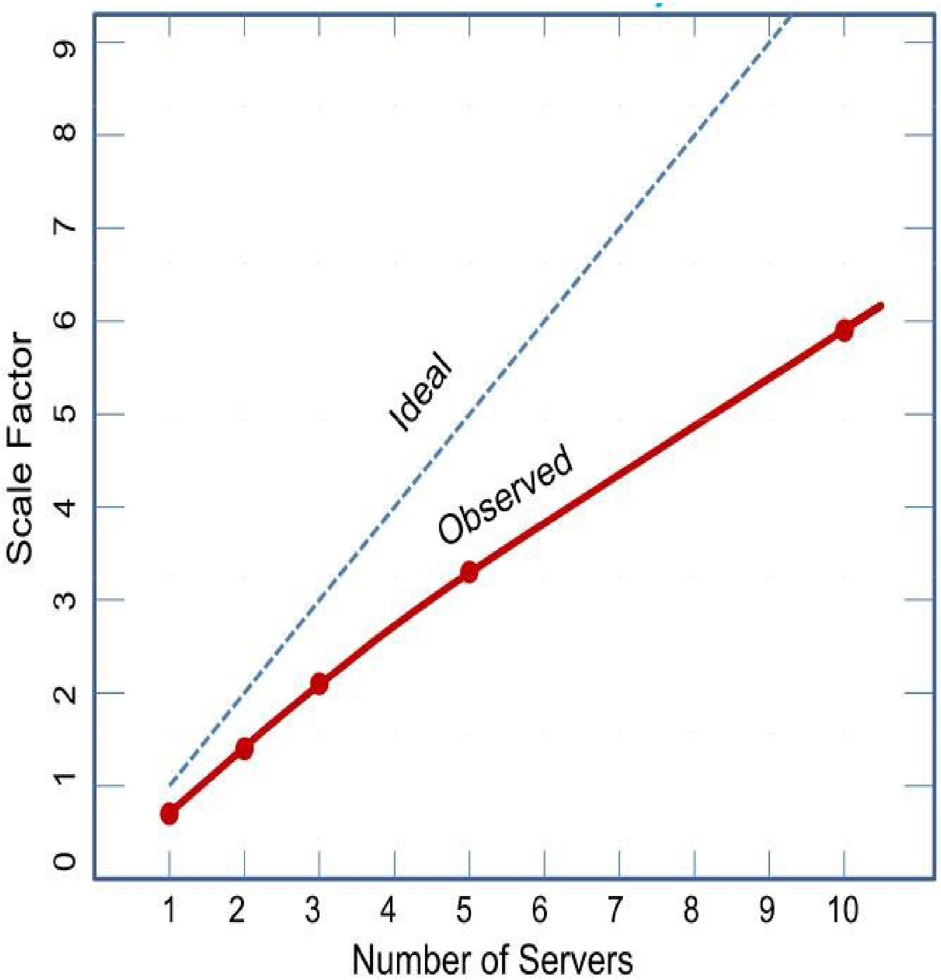
Example DBT-1 (2)

- customer, address, orders, order_line, cc_xacts
  - Frequently written
    - hence distributed
  - Participate in JOINs amongst each other with customer_id as JOIN key
  - point SELECTs based on customer_id
    - hence diistributed by hash on customer_id so that JOINs are shippable
  - Participate in JOINs with item
    - Having item replicated helps pushing JOINs to datanode

Example DBT-1 (3)

- Shopping_cart, shopping_cart_line
  - Frequently written
    - Hence distributed
  - Point selects based on column shopping_cart_id
    - Hence distributed by hash on shopping_cart_id
  - JOINs with item
    - Having item replicated helps

DBT-1 scale-up



- Old data, we will publish bench-marks for 1.0 soon.
- DBT-1 (TPC-W) benchmark with some minor modification to the schema
- 1 server = 1 coordinator + 1 datanode on same machine
- Coordinator is CPU bound
- Datanode is I/O bound

Other scaling tips

Using GTM proxy

- GTM can be a bottleneck
  - All nodes get snapshots, transactions ids etc. from GTM
- GTM-proxy helps reduce the load on GTM
  - Runs on each physical server
  - Caches information about snapshots, transaction ids etc.
  - Serves logical nodes on that server

Adding coordinator and datanode

- Coordinator
  - Scaling connection load
  - Too much load on coordinator
  - Query processing mostly happens on coordinator
- Datanode
  - Data scalability
    - Number of tables grow – new nodes for new tables/databases
    - Distributed table sizes grow – new nodes providing space for additional data
  - Redundancy

Impact of transaction management on performance

- 2PC is used when
  - More than one node performs write in a transaction
  - Explicit 2PC is used
  - More than one node performs write during a single statement
- Only nodes performing writes participate in 2PC
- Design transactions such that they span across as few nodes as possible.

DBT-2 (sneak peek)

- Like TPC-C
- Early results show 4.3 times scaling with 5 servers
  - More details to come ...

Thank you

ashutosh.bapat@enterprisedb.com