Scaling out by distributing and replicating data in Postgres-XC

Ashutosh Bapat
@Postgres Open 2012
Agenda

- What is Postgres-XC
- Postgres-XC architecture overview
- 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

- **Synchronous**
  - Writes to one node are reflected on all the nodes

- **Transparent**
  - Applications need not care about the data distribution
Postgres-XC architecture

- **Coordinators**: Add coordinators
- **Datanodes**: Add datanodes
- **SQL + libpq interface**: SQL statements from applications

**Postgres-XC cluster**

- **GTM**: Transaction info
- **SQL + libpq interface**
- **Add coordinators**
- **Add datanodes**

The Enterprise PostgreSQL Company
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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>val2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Writes**
- write
- write
- write

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>val2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reads**
- read

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>val2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

**Write**

```
<table>
<thead>
<tr>
<th>val</th>
<th>val2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
```

**Read Combiner**

```
<table>
<thead>
<tr>
<th>val</th>
<th>val2</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>101</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>31</td>
<td>41</td>
</tr>
<tr>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>21</td>
<td>101</td>
</tr>
<tr>
<td>31</td>
<td>41</td>
</tr>
<tr>
<td>30</td>
<td>40</td>
</tr>
</tbody>
</table>
```

**write**

**read read read**
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 occurring 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
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 distributed 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 benchmarks 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