Schism: a Workload-Driven Approach to Database Replication and Partitioning

Carlo Curino et al.
Presented by: Rui Zhang
Motivation

• Why partitioning?
  • Scalability
  • Availability
  • Manageability

• Traditional Partitioning
  • Total replication
  • Round-Robin
  • Range
  • Hash
Motivation (Cont.)

• Distributed transactions are expensive  
  • Consensus protocol  
  • Distributed locks  
  • More communication

• Many to Many relationship  
  • Hard to partition
Solution

• Minimize distributed transactions

• Balance the workload of each partition
Schism

• Part of RelationalCloud System
  • An MIT-based effort to investigate technologies and challenges related to Database-as-a-Service (DAAS) within cloud-computing[1]
  • Researchers including Carlo, Evan and Sam participate this project

• DAAS
  • Cloud computing based service is a novel market for data management
  • More and more dedicated data management engines which complicates the process of choosing and deploying
  • DAAS is to hide the complexity with a simple interface

Schism

• Input
  • Workload trace

• Output
  • Partitioning and replication plan

• Five steps
  • Data pre-processing
  • Creating the graph
  • Partitioning the graph
  • Explaining the partition
  • Final Validation
Creating the graph

• Node: Tuple
• Edge: Transaction
• Weight: Number of transactions
Creating the graph

<table>
<thead>
<tr>
<th>account</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

...  .....  ....
BEGIN
UPDATE account SET bal=60k
WHERE id=2;
SELECT * FROM account
WHERE id=5;
COMMIT

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>bal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>carlo</td>
<td>80k</td>
</tr>
<tr>
<td>2</td>
<td>evan</td>
<td>60k</td>
</tr>
<tr>
<td>3</td>
<td>sam</td>
<td>129k</td>
</tr>
<tr>
<td>4</td>
<td>eugene</td>
<td>29k</td>
</tr>
<tr>
<td>5</td>
<td>yang</td>
<td>12k</td>
</tr>
</tbody>
</table>

transaction edges
Creating the graph

BEGIN
UPDATE account SET bal=60k WHERE id=2;
SELECT * FROM account WHERE id=5;
COMMIT

BEGIN
UPDATE account SET bal=bal-1k WHERE name="carlo";
UPDATE account SET bal=bal+1k WHERE name="evan";
COMMIT

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>bal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>carlo</td>
<td>80k</td>
</tr>
<tr>
<td>2</td>
<td>evan</td>
<td>60k</td>
</tr>
<tr>
<td>3</td>
<td>sam</td>
<td>129k</td>
</tr>
<tr>
<td>4</td>
<td>eugene</td>
<td>29k</td>
</tr>
<tr>
<td>5</td>
<td>yang</td>
<td>12k</td>
</tr>
<tr>
<td>...</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
Creating the graph

BEGIN
UPDATE account SET bal=bal-1k WHERE name="carlo";
UPDATE account SET bal=bal+1k WHERE name="evan";
COMMIT

BEGIN
UPDATE account SET bal=60k WHERE id=2;
SELECT * FROM account WHERE id=5;
COMMIT

BEGIN
UPDATE account SET bal=bal-1k WHERE name="carlo";
UPDATE account SET bal=bal+1k WHERE name="evan";
COMMIT

BEGIN
SELECT * FROM account WHERE id IN {1,3}
ABORT

---

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>bal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>carlo</td>
<td>80k</td>
</tr>
<tr>
<td>2</td>
<td>evan</td>
<td>60k</td>
</tr>
<tr>
<td>3</td>
<td>sam</td>
<td>129k</td>
</tr>
<tr>
<td>4</td>
<td>eugene</td>
<td>29k</td>
</tr>
<tr>
<td>5</td>
<td>yang</td>
<td>12k</td>
</tr>
<tr>
<td>...</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
BEGIN
UPDATE account SET bal=bal-1k WHERE name="carlo";
UPDATE account SET bal=bal+1k WHERE name="evan";
COMMIT

BEGIN
UPDATE SET bal=bal+1k WHERE bal < 100k;
COMMIT

BEGIN
SELECT * FROM account
WHERE id IN {1,3}
ABORT

BEGIN
UPDATE account SET bal=60k WHERE id=2;
SELECT * FROM account
WHERE id=5;
COMMIT

BEGIN
UPDATE account SET bal=bal-1k WHERE name="carlo";
UPDATE account SET bal=bal+1k WHERE name="evan";
COMMIT

BEGIN
SELECT * FROM account
WHERE id IN {1,3}
ABORT

BEGIN
UPDATE SET bal=bal+1k WHERE bal < 100k;
COMMIT

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>bal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>carlo</td>
<td>80k</td>
</tr>
<tr>
<td>2</td>
<td>evan</td>
<td>60k</td>
</tr>
<tr>
<td>3</td>
<td>sam</td>
<td>129k</td>
</tr>
<tr>
<td>4</td>
<td>eugene</td>
<td>29k</td>
</tr>
<tr>
<td>5</td>
<td>yang</td>
<td>12k</td>
</tr>
<tr>
<td>...</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

transaction edges
Creating the graph

BEGIN
UPDATE account SET bal=bal-1k WHERE name="carlo";
UPDATE account SET bal=bal+1k WHERE name="evan";
COMMIT

BEGIN
UPDATE account SET bal=bal+1k WHERE bal < 100k;
COMMIT

BEGIN
SELECT * FROM account
WHERE id IN {1,3}
ABORT

BEGIN
UPDATE account SET bal=60k
WHERE id=2;
SELECT * FROM account
WHERE id=5;
COMMIT

BEGIN
UPDATE account SET bal=bal-1k WHERE name="carlo";
UPDATE account SET bal=bal+1k WHERE name="evan";
COMMIT

BEGIN
SELECT * FROM account
WHERE id IN {1,3}
ABORT

BEGIN
UPDATE account SET bal=bal+1k
WHERE bal < 100k;
COMMIT

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>bal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>carlo</td>
<td>80k</td>
</tr>
<tr>
<td>2</td>
<td>evan</td>
<td>60k</td>
</tr>
<tr>
<td>3</td>
<td>sam</td>
<td>129k</td>
</tr>
<tr>
<td>4</td>
<td>eugene</td>
<td>29k</td>
</tr>
<tr>
<td>5</td>
<td>yang</td>
<td>12k</td>
</tr>
<tr>
<td>...</td>
<td>.....</td>
<td>.....</td>
</tr>
</tbody>
</table>
Tuple Replication
Tuple Replication

<table>
<thead>
<tr>
<th>tuple id</th>
<th>partition label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Graph Partitioning

• Metis algorithm
  • Coarsening
  • Initial partitioning
  • Refinement
See www.cs.umn.edu/~metis for more detail

• Goal
  • Minimum cut and balanced partitions

• Output
  • Lookup table
Graph Size Reduction

• Sampling
  • Transaction level
  • Tuple level
• Blanket-statement filtering
• Relevance filtering
• Star-shaped replication
• Tuple coalescing
Explanation Phase

• Method
  • Decision Tree

• Input
  • Tuple->partition mappings

• Output
  • Compact Model

<table>
<thead>
<tr>
<th>tuple id</th>
<th>partition label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

(id = 1) -> partitions = {0, 1}
(2 <= id < 4) -> partition = 0
(id >= 4) -> partition = 1
Explanation Phase

• Candidate attributes
  • Frequent used attributes in WHERE clauses
    • Item ids and warehouse ids for “stock”
  • Join predicates
    • Co-location of the joined tuples
    • Limited use (perform the join) but supported
Query/Tuple Routing

• Centralized Router
  • A decision tree
  • Determine which partition the query or tuple belongs to
  • Spare partition that store tuples don’t hit
    • If full, rerun the partitioning algorithm and retrain the decision tree
    • After retraining, some tuples should be migrated from one partition to another via data migration SQL scripts
  • Broadcast to all partitions for attributes that are not partitioning attributes
Evaluation

• Schism
• Manual
• Total Replication
• Hash partitioning
Evaluation

<table>
<thead>
<tr>
<th>dataset:</th>
<th>YCSB-A</th>
<th>YCSB-E</th>
<th>TPCC-2W</th>
<th>TPCC-2W</th>
<th>TPCC-50W</th>
<th>TPC-E</th>
<th>EPINIONS</th>
<th>EPINIONS</th>
<th>RANDOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>partitions:</td>
<td>any</td>
<td>100</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>coverage:</td>
<td>any</td>
<td>1%</td>
<td>50%</td>
<td>0.5%</td>
<td>1%</td>
<td>5%</td>
<td>15%</td>
<td>15%</td>
<td>any</td>
</tr>
<tr>
<td>SCHISM:</td>
<td>hashing</td>
<td>range-predicates</td>
<td>range-predicates</td>
<td>range-predicates</td>
<td>range-predicates</td>
<td>look-up table</td>
<td>look-up table</td>
<td>hashing</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Node Count</th>
<th>Edge Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-E (3M nodes; 100M edges)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPC-C 50W (2.5M nodes; 65M edges)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Epinions.com (600k nodes; 5M edges)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partitioning time (sec) vs. number of partitions.
Evaluation

The graph shows the throughput (txns/s) as a function of partitions. Two lines are plotted:
- The solid line represents 16 warehouses / machine.
- The dashed line represents 16 warehouses total.

As the number of partitions increases, the throughput also increases linearly for both cases.
Limitation

• Not suitable for insert-heavy workloads
  • Involves many data migrations

• Not suitable for databases with large size
  • Partitioning time will be long

• There will be false positive in decision tree
  • Overhead to correct this error

• Compared with Horticulture, it does not consider temporal skew