Database Design

Wenfeng Xu
Hanxiang Zhao
Automated Partitioning Design in Parallel Database Systems

- MPP system:
- A distributed computer system which consists of many individual nodes, each of which is essentially an independent computer in itself.
• Bottleneck: Excessive data transfers
• How to cope?
• Originally partitioned in an adequate way
Two categories:
1) Optimizer-independent
2) Shallowly-integrated
Two problems:
1) recommendations suffer from the tuning tools not being in-sync with optimizer's decisions
2) performance of the tuning tool is likely to dimish due to narrow APIs between the tool and the DBMS
• Advisor:
• Deeply-integrated
• Parallel query optimizer.
• PDW: appliance
• Plan Generation and Execution

Figure 4: Parallel query optimization flow (all on the control node).
Query plan->parallel execution plan(DSQL)

DSQL:
1) SQL operations
   an SQL statement to be executed against
   the underlying compute node’s DBMS instance
2) Data movement operations
   transfer data between DBMS instances on
   different nodes
Figure 3: Parallel query optimization flow: (a) input query, (b) logical query tree, (c) augmented MEMO, (d) best query plan, (e) final DSQL plan.

Example: Consider the following SQL query:

```
SELECT *
FROM CUSTOMER C, ORDERS O
WHERE C.C_CUSTKEY = O.O_CUSTKEY
AND O.O_TotalPrice > 10000
```
• MEMO: recursive data structure
• Groups and groupExpressions
AUTOMATED PARTITIONING DESIGN

PROBLEM

Given a database $D$, a query workload $W$, and a storage bound $B$, find a partitioning
strategy (or configuration) for $D$ such that

(i) the size of replicated tables fits in $B$, and

(ii) the overall cost of $W$ is minimized.
Figure 5: Partitioning advisor architecture.
• the complex search space
• the search algorithm
• the evaluation mechanism
- shallowly-integrated approach for partitioning tuning design:
  - 1) Rank-Based Algorithm
  - 2) Generic Algorithm

![Flowchart for genetic algorithm.](figure6.png)
• \{nation, supplier, region, lineitem, orders, \\
partsupp, \\
customer, part\} \rightarrow \\
\{R,R,R,D1,D2,D1,D1,D1,D1\},
Disadvantage of Shallowly-Integrated Approaches

1) search space is likely to be extremely large
2) each evaluation of a partitioning configuration is expensive
TUNING WITH DEEP OPTIMIZER
INTEGRATION
MESA
“workload memo”
Figure 7:
Interesting Columns
1) columns referenced in equality join
2) any subset of group-by columns
*-partitioning:
“every” partition or replication option for a base table is simultaneously available
Branch and Bound Search
Pruning: discards subtrees when a node or any of its descendants will never be either feasible or optimal
- Figure 8
- Node, Leaf, Bud, Bounding function,
- Incumbent
- 1) Node selection policy
- 2) Table/column selection policy
- 3) Pruning strategy
- 4) Bud node promotion
- 5) Stopping condition

Figure 8: Branch and bound enumeration tree for partitioning configuration search problem.
MESA Algorithm

```python
MESA (W: workload, B: storage bound)
01 wMemo = CreateWorkloadMemo(W, B)
02 incumbent = null
03 bbTree = CreateRoot(wMemo)
04 while (!stop_condition())
05     currConfig = SelectNode(bbTree) // DFS policy
06     newConfig = CreateChildConfig(currConfig) // table/column selection policy
07     if (newConfig violates B constraint)
08         prune(newConfig)
09     else
10         cost = ParallelPostProcess(wMemo, newConfig)
11         if (newConfig is leaf or can be promoted)
12             if (cost < incumbent.cost)
13                 incumbent = newConfig
14                 prune(newConfig)
15         else // partially defined configuration
16             if (incumbent.cost < cost)
17                 prune(newConfig)
18     return incumbent
```

Figure 9: Memo-based search algorithm using branch and bound enumeration.
• Experimental Evaluation
• Table 1,2,3
• We compare the quality of the recommendations produced by each technique
### Table 1: Experimental benchmarks

<table>
<thead>
<tr>
<th>Benchmark (scale)</th>
<th># Tables</th>
<th>Workload (# queries)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-H (1TB)</td>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>TPC-DS (1TB)</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>L’Oreal (88GB)</td>
<td>573</td>
<td>29</td>
</tr>
<tr>
<td>MSSales (800GB)</td>
<td>346</td>
<td>27</td>
</tr>
</tbody>
</table>

### Table 2: GA parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td># of generations</td>
<td>100</td>
<td># of times the population will be replaced through reproduction.</td>
</tr>
<tr>
<td>Population size</td>
<td>30</td>
<td># of chromosomes available for use during the search.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the size is too big, GA will spend unnecessarily long time evaluating</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chromosomes, if it is too small, GA may have no chance to adequately</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cover the search space.</td>
</tr>
<tr>
<td>Crossover rate</td>
<td>0.1</td>
<td>the probability of crossover between two chromosomes.</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>0.1</td>
<td>the probability that values of genes of a newly created (or selected)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>off-springs will be randomly changed.</td>
</tr>
<tr>
<td>Selection rate</td>
<td>0.2</td>
<td>the percentage of the worst of the current population that will be discarded</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(after re-generation).</td>
</tr>
</tbody>
</table>

### Table 3: MESA parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node selection</td>
<td>DFS</td>
<td>the forward- and the back-tracking policy in the branch and bound tree.</td>
</tr>
<tr>
<td>Variable selection</td>
<td>replicate, distribute by rank</td>
<td>See Section 5.5 for details.</td>
</tr>
<tr>
<td>Stop condition</td>
<td>150</td>
<td>the number of iterations after which the search terminates.</td>
</tr>
</tbody>
</table>
Table 4: Comparison of techniques

<table>
<thead>
<tr>
<th>Approach</th>
<th>Quality (Q)</th>
<th>MESA (sec)</th>
<th>Total Time</th>
<th>MESA (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-H</td>
<td>RANK</td>
<td>1.4</td>
<td>6 min 46 sec</td>
<td>7.71</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>1.3</td>
<td>6 min 58 sec</td>
<td>40.56</td>
</tr>
<tr>
<td></td>
<td>MESA</td>
<td>1.1</td>
<td>3 min 52 sec</td>
<td>78.38</td>
</tr>
<tr>
<td>TPC-DS</td>
<td>RANK</td>
<td>1.1</td>
<td>5 min 27 sec</td>
<td>51.3</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>1.0</td>
<td>5 min 52 sec</td>
<td>51.76</td>
</tr>
<tr>
<td></td>
<td>MESA</td>
<td>1.1</td>
<td>3 min 32 sec</td>
<td>46.72</td>
</tr>
<tr>
<td>L’Oreal</td>
<td>RANK</td>
<td>1.2</td>
<td>8 min 12 sec</td>
<td>101.4</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>1.2</td>
<td>8 min 12 sec</td>
<td>101.4</td>
</tr>
<tr>
<td></td>
<td>MESA</td>
<td>1.2</td>
<td>5 mins 1 sec</td>
<td>101.4</td>
</tr>
<tr>
<td>MSSales</td>
<td>RANK</td>
<td>1.1</td>
<td>6 min 8 sec</td>
<td>42.78</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>1.2</td>
<td>7 min 58 sec</td>
<td>51.3</td>
</tr>
<tr>
<td></td>
<td>MESA</td>
<td>1.2</td>
<td>5 mins 38 sec</td>
<td>51.3</td>
</tr>
</tbody>
</table>

Figure 10: Quality over time: TPC-H.

Figure 11: Quality over time: TPC-DS.

Figure 12: Quality over time: L’Oreal.

Figure 13: Quality over time: MSSales.
Impact of replication bound

Figure 14: Quality of recommendations under various replication bounds.
• Performance of MESA
• Workload MEMO construction overhead

Figure 15: Time overhead of workload MEMO creation.
Subsequent reoptimization calls
EXTENSIONS
Updates
Multi-Column Partitioning
Range Partitioning
Interaction With Other Physical Design
Structures
CONCLUSION

- techniques for finding the best partitioning configuration in distributed environments
- deep integration with the parallel query optimizer
- Using its internal MEMO data structure for faster evaluation of partitioning configurations and to provide lower bounds during a branch and bound search strategy
Schism: a Workload-Driven Approach to Database Replication and Partitioning
Problem: distributed transactions are expensive in OLTP settings. why: two-phase commit

Solution: minimize the number of distributed transactions, while producing balanced partitions.

Introduce: Schism
H-store
Schism

- Five steps:
  - Data pre-processing
  - Creating the graph
  - Partitioning the graph
  - Explaining the partition
  - Final validation
Graph Representation

- notion: node, edge, edge weights
- example: a bank database (from paper)
- workload: 4 transactions

Figure 2: The graph representation
Graph Representation

- an extension of the basic graph representation
- Graph replication: “exploding” the node representing a single tuple into a star-shaped configuration of \( n + 1 \) nodes. (Figure 3 from paper)
Graph Partitioning

- split graph into k partitions $\rightarrow$ overall cost of the cut edges is minimized.
- result: a fine-grained partition
- lookup table: node--partition label
- note: replicated tuple

![Graph with replication](image)

**Figure 3: Graph with replication**
Explanation Phase

- use decision tree to find a compact model that captures the (tuple, partition) mappings.
- \((id = 1) \rightarrow \text{partitions} = \{0, 1\}\)
- \((2 \leq id < 4) \rightarrow \text{partition} = 0\)
- \((id \geq 4) \rightarrow \text{partition} = 1\)
Final Validation

• compare solutions to select the final partitioning scheme.
• fine-grained per-tuple partitioning, range-predicate partitioning, hash-partitioning
Optimization

• graph partitioners scale well in terms of the number of partitions, but running time increases substantially with graph size.

• methods for reducing size of graph:
  transaction-level sampling
  tuple-level sampling
  tuple-coalescing
Experimental Evaluation

Figure 4: Schism database partitioning performance.
Thank you!