The main design philosophy is that no single data structure/algorithm wins. Thus from the high-level perspective we build a system with clearly abstracted modules to address this complexity of supporting a rich collection of indexes, specialized operators, and parallel execution strategies.

**System Architecture:**

- Data
- Query
- Preprocessor
- Statistics & Indexes
- Compressed Storage
- Optimizer
- Execution Generator
- Query Evaluation Main Task
- Inter-query parallelism
- Inter-operator parallelism
- Intra-operator parallelism
- Batch Start
- Batch Complete
- Evaluate-One Query
- Parse
- Optimize
- Generate Execution Plan
- Aggregate Operation
- Print Tuple Operation
- Aggregate Finalization
- Print Results

**Key Mechanisms:**

- No “silver bullet” operator/index -- need to implement a rich collection of them and design proper rules to choose.
- Apply aggressive operator fusion to avoid the cost of materializing intermediate results.

**Testing machine configuration:**

2x Intel Xeon E5-2660 v2 (2.2 GHz), 20 cores / 40 hyperthreads, 256 GB DDR3 RAM

**Task:** Evaluate as fast as possible batches of SPJA (Selection-Projection-Join-Aggregation) queries on a set of immutable relations.

Each query involves up to 4 relations. Aggregate functions are always SUM without GROUP BY.

**1. Contest Overview**

- Use existence maps to figure out containment relationship among columns.
- Build various indexes on each relation (primary key index, foreign key index, count vector index) based on relation size.

**2. Our Approach**

The main design philosophy is that no single data structure/algorithm wins. Thus from the high-level perspective we build a system with clearly abstracted modules to address this complexity of supporting a rich collection of indexes, specialized operators, and parallel execution strategies.

**3. Preprocessing**

- Calculate min/max values of each column. Meanwhile compress the column if possible (simply truncating the leading zeros).
- Build existence bitmap for each column, and use it to count the number of unique values in the column.

**4. Optimizer**

- ~20 optimization passes.
- Some of the optimization rules: filter pushdown, early projection, range propagation, predicate simplification, semi-join elimination, common aggregate-expression elimination.
- Heuristic-based join order optimization.
- Identify the shape of multi-relation joins (multi-way join, linear join, star join) and apply corresponding specialized operators.

**5. Execution**

- Build a scheduler that supports dynamic task DAG spawning to fully utilize CPU resources.
- Implement a collection of relational operators. Some regular operators are: select, hash join, sort merge join, index scan, index lookup join.
- When applicable, fuse multiple binary joins and the top-level aggregation into one multi-relation join-aggregate operator. Example operators are multi-way-join-aggregate and linear-join-aggregate.

**6. Contest Workload and Results**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
<th>X-Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>9.4 MB</td>
<td>89.5 MB</td>
<td>3.9 GB</td>
<td>6.6 GB</td>
</tr>
<tr>
<td># Relations</td>
<td>14</td>
<td>12</td>
<td>29</td>
<td>34</td>
</tr>
<tr>
<td># Queries</td>
<td>50</td>
<td>146</td>
<td>50</td>
<td>146</td>
</tr>
<tr>
<td># Batches</td>
<td>5</td>
<td>21</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>Execution Time (s)</td>
<td>0.027</td>
<td>0.133</td>
<td>0.547</td>
<td>1.475</td>
</tr>
</tbody>
</table>

*This work was supported in part by CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and grant FA8650-15-C-7562 also from DARPA.*