Special Topics:
Self-Driving Database Management Systems

Index Recommendation III

@DJ_Mooshoo// 15-799 // Spring 2022
TODAY’S AGENDA

Overview
Index Selection Algorithms
Methods
Results
Parting Thoughts
TODAY’S AGENDA

Overview
• Terminology
• Motivation
• Challenges

Index Selection Algorithms

Methods

Results

Parting Thoughts
TERMINOLOGY

Workload and Index
• Simplifying observation: define by attributes

Index Configuration

Potential Index

Syntactically Relevant

Index Candidates

Index Interaction
• Can be positive or negative [0]
Problem: There are many approaches to index selection, but comparisons between algorithms is limited.

Goal: compare state-of-the-art index selection algorithms more comprehensively by:

• Measuring in multiple dimensions
• Developing a standard framework for comparisons
CHALLENGES

• Different Goals
  • Maximize Benefit or Benefit/Storage

• Algorithms with Parameters
  • Choosing the right setting for a workload

• Query Cost Estimation
  • DBMS - specific
  • Often not reflective of actual cost
THIS PAPER

• Survey of 8 index selection algorithms
• Provides an evaluation framework that addresses the challenges (in part)
• Present evaluations of the algorithms within the framework
TODAY’S AGENDA

Overview
Index Selection Algorithms
Methods
Results
Parting Thoughts
OVERVIEW OF ALGORITHMS

8 algorithms varying in:

• Approach
• Objective and stop criteria
• Academic/Commercial/Open Source
• Complexity (?)
1. Start with *every* single-column index
2. Drop index that leads to the lowest cost of workload
3. Stop when cost cannot be reduced

Original version uses own cost model

Modifications:
Use the framework for cost estimation
Stop when maximum # indexes is reached
AutoAdmin (1997)

Microsoft SQL Server Tuner

1. Start with per-query candidates
2. Naïve Enumeration
3. Greedy extension
   1. Adding indexes
   2. Adding columns to indexes
4. Stop at maximum # index

Reduce estimation calls using “atomic configurations”
Core approach is the same as AutoAdmin, plus:

1. Also tunes materialized views and partitioning (not evaluated)
2. Considers multi-column indexes from the start
3. Merges query-level candidates
4. Considers index interaction to avoid evaluating suboptimal sets
5. Stop at any time
DB2 ADVISOR (2000)

1. All candidates from every query added as hypothetical index
2. Indexes which are used by optimizer added to candidate set
3. Sort candidates by benefit-per-space ratio
4. Randomly vary set to account for index interaction
1. Start with optimal index set for each query
   → Original paper exploits optimizer code paths
2. Union of all query-level sets to create (huge) candidate set
3. Reduce candidate set (relaxing) by iteratively:
   → Merging
   → Removing attributes (Prefixing)
   → Promote to
   →Removing Indexes
CoPhy LP (2011)

1. Formulate index selection as an integer linear program
2. Use off-the-shelf solver to find optimal solution
3. Scalability Issues
   → Binary variables for each (index, query) pair
   → A variable for each subset of candidate set
   → Solution: “Decomposition Heuristic” to reduce problem size [1]
1. Gather queries and runtime information, templatize them
2. Add hypothetical indexes of all single and 2-column index to configuration
3. Run explain to see which indexes are chosen – use these
1. Start with an empty solution set
2. Greedily pick action with the greatest reduction in cost/storage
   • Adding a new index
   • Appending an attribute to an existing index
3. Stop when no cost reduction can be made or storage budget is met
SUMMARY

• Query-based (DB2Advis and Dexter) vs Index combination-based
  • Speed vs index interactions
• Approach
  • Additive (AutoAdmin, DTA, Extend)
  • Reductive (Drop, Relaxation)
TODAY’S AGENDA

Overview
Index Selection Algorithms
Methods
  • Benchmarks
  • Framework
  • Evaluation
Results
Parting Thoughts
## Benchmarks

**Table 2:** Metrics for the evaluated benchmark schemata and workloads. The number of relevant index candidates was determined by generating all permutations of all syntactically relevant indexes.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Relations</th>
<th>Attributes</th>
<th>Queries</th>
<th>Relevant n-column candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB</td>
<td>Real-world</td>
<td>21</td>
<td>108</td>
<td>113</td>
<td>73 218 552 1080</td>
</tr>
<tr>
<td>TPC-H</td>
<td>Synthetic uniform</td>
<td>8</td>
<td>61</td>
<td>22</td>
<td>53 398 3306 29088</td>
</tr>
<tr>
<td>TPC-DS</td>
<td>Synthetic skewed</td>
<td>24</td>
<td>429</td>
<td>99</td>
<td>248 3734 68052 1339536</td>
</tr>
</tbody>
</table>

- **TPC-H (10x)** – Relatively small OLAP benchmark
- **TPC-DS (10x)** – More sophisticated OLAP benchmark
- **Join Order Benchmark** – based on IMDB
  - Queries focused on joins, not a lot of wide column indices
- No writes/updates – purely analytical
  - No index maintenance cost
FRAMESWORK

• PostgreSQL 12 chosen due to HypoPG
• Algorithms reimplemented in Python 3
• Key Concept: Abstraction Layers
  • CostEvaluation
  • DatabaseConnector
• Cost Estimation Caching
Solution quality with respect to storage constraint
• Cost reduction
• Algorithm Runtime
• Solution Granularity

Potential Indexes
• All (relevant) indexes of width 2
TODAY’S AGENDA

Overview
Index Selection Algorithms
Methods
Results
• Per Benchmark
• Further Dimensions
• Important Findings
Parting Thoughts
### TPC-H

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Relations</th>
<th>Attributes</th>
<th>Queries</th>
<th>Relevant $n$-column candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-H</td>
<td>Synthetic uniform</td>
<td>8</td>
<td>61</td>
<td>22</td>
<td>(n = 1) 53 (n = 2) 398 (n = 3) 3306 (n = 4) 29088</td>
</tr>
</tbody>
</table>

- Stop criteria leads to differences in solution characteristic
  - **AutoAdmin** and Drop only find solution at 2GB, but it’s a good one
  - Due to a “dominating table”
- Best solution depends on storage budget
Stop criteria leads to differences in solution characteristic
- **AutoAdmin** and **Drop** only find solution at 2GB, but it’s a good one
- Due to a “dominating table”

Best solution depends on storage budget
• Extend and DTA are best when budget < 6GB
  • Additive approach
• Extend and Relaxation find best solutions
  • Not the fastest
• Dexter has poor granularity

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Relations</th>
<th>Attributes</th>
<th>Queries</th>
<th>Relevant $n$-column candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-DS</td>
<td>Synthetic skewed</td>
<td>24</td>
<td>429</td>
<td>99</td>
<td>$n = 1$ 248 $n = 2$ 3734 $n = 3$ 68052 $n = 4$ 1339536</td>
</tr>
</tbody>
</table>

![Graph showing relative workload cost vs. index storage consumption]
• Extend and DTA are best when budget < 6GB  
  • Additive approach
• Extend and Relaxation find best solutions  
  • Not the fastest
• Dexter has poor granularity
TPC-DS

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Relations</th>
<th>Attributes</th>
<th>Queries</th>
<th>Relevant n-column candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-DS</td>
<td>Synthetic skewed</td>
<td>24</td>
<td>429</td>
<td>99</td>
<td>248</td>
</tr>
</tbody>
</table>

- Extend and DTA are best when budget < 6GB
  - Additive approach
- Extend and Relaxation find best solutions
  - Not the fastest
- Dexter has poor granularity
**TPC-DS**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Relations</th>
<th>Attributes</th>
<th>Queries</th>
<th>Relevant $n$-column candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-DS</td>
<td>Synthetic skewed</td>
<td>24</td>
<td>429</td>
<td>99</td>
<td>$n = 1$ 248 $n = 2$ 3734 $n = 3$ 68052 $n = 4$ 1339536</td>
</tr>
</tbody>
</table>

- DB2Advis is fast
  - Because it only calls CostEstimation on 2 configurations (“all” and “none”)
- Relaxation scales poorly with number of potential indexes
  - Compared to TPC-H
**TPC-DS**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Relations</th>
<th>Attributes</th>
<th>Queries</th>
<th>Relevant $n$-column candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-DS</td>
<td>Synthetic skewed</td>
<td>24</td>
<td>429</td>
<td>99</td>
<td>248, 3734, 68052, 1339536</td>
</tr>
</tbody>
</table>

- **DB2Advis is fast**
  - Because it only calls CostEstimation on 2 configurations (“all” and “none”)
- **Relaxation scales poorly with number of potential indexes**
  - Compared to TPC-H
**JOIN ORDER BENCHMARK**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Relations</th>
<th>Attributes</th>
<th>Queries</th>
<th>Relevant $n$-column candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB</td>
<td>Real-world</td>
<td>21</td>
<td>108</td>
<td>113</td>
<td>$n = 1$ 73 $n = 2$ 218 $n = 3$ 552 $n = 4$ 1080</td>
</tr>
</tbody>
</table>

- Workload lends itself to massive speedups
  - Adding relatively small
- Dexter has poor granularity
JOIN ORDER BENCHMARK

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Relations</th>
<th>Attributes</th>
<th>Queries</th>
<th>Relevant $n$-column candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB</td>
<td>Real-world</td>
<td>21</td>
<td>108</td>
<td>113</td>
<td>$n = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$n = 2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$n = 3$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$n = 4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>73</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>218</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>552</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1080</td>
</tr>
</tbody>
</table>

- Reduction vs Additive Approach
  - Runtime decreases/increases with respect to storage budget
  - Initial candidate set has large impact on runtime
INFLUENCE OF PARAMETERS

- **DB2Advis – Try Variation**
  - Candidate set too large for random variation to reliably find improvements
  - Could be helpful in databases with fewer tables/attributes

- **AutoAdmin – Naïve Enumerations**
  - Increasing $k=1$ to 2 increases runtime 3-10x
  - Sometimes smaller $k$ leads to better solution

- **DTA – Runtime Limits**
  - Running **9 minutes** leads to a solution within 3% of running **14 hours**

*These may be dependent on workload and DBMS*
**FURTHER DIMENSIONS**

- Index selection order
  - For a specific algorithm and workload, what indexes are selected - and when?
- Fine-grain analysis of algorithm
- Runtime cost breakdown

Figure 4: Estimated query processing costs for TPC-H (scale factor 10) on PostgreSQL. Queries 1, 3, 6, 7, 10, 13, 14, 15, and 16 are omitted as their costs were not affected by indexes for a budget of 5 GB. Expensive queries (2, 17, 20) depicted with log (right), others (left) with linear scale. S is the final index configuration.
FURTHER DIMENSIONS

- Cost Request Caching
  - Although the cache is an implementation detail, it does allow us to obtain useful information about what index configurations each algorithm tries

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Configurations</th>
<th>Index simulations</th>
<th>Cost requests</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Total</td>
<td>Non-cached</td>
</tr>
<tr>
<td>AutoAdmin</td>
<td>129</td>
<td>10,991</td>
<td>33,851</td>
<td>11,676</td>
</tr>
<tr>
<td>Naive-2</td>
<td>816</td>
<td>73,504</td>
<td>240,441</td>
<td>73,440</td>
</tr>
<tr>
<td>CoPhy</td>
<td>3,983</td>
<td>3,982</td>
<td>394,317</td>
<td>52,177</td>
</tr>
<tr>
<td>DB2Advis</td>
<td>2</td>
<td>7,179</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>DTA</td>
<td>1,442</td>
<td>25,812</td>
<td>1,650,510</td>
<td>129,811</td>
</tr>
<tr>
<td>Dexter</td>
<td>2</td>
<td>3,982</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>Drop</td>
<td>203</td>
<td>29,144</td>
<td>2,601,450</td>
<td>18,348</td>
</tr>
<tr>
<td>Extend</td>
<td>594</td>
<td>11,295</td>
<td>812,430</td>
<td>53,472</td>
</tr>
<tr>
<td>Relaxation</td>
<td>1,898</td>
<td>51,680</td>
<td>2,982,690</td>
<td>170,863</td>
</tr>
</tbody>
</table>
• Index width threshold
  • Extend is the only algorithm that can handle \( w \geq 4 \)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dataset</th>
<th>Relations</th>
<th>Attributes</th>
<th>Queries</th>
<th>Relevant n-column candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC-DS</td>
<td>Synthetic skewed</td>
<td>24</td>
<td>429</td>
<td>99</td>
<td>248 3734 68052 1339536</td>
</tr>
</tbody>
</table>

**Table 4:** Cost request timings including index simulation for two TPC-DS queries; DNF exceeds 30min.

<table>
<thead>
<tr>
<th>Index width</th>
<th>Relevant indexes</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Query 13</td>
<td>Query 64</td>
</tr>
<tr>
<td>1 column</td>
<td>22</td>
<td>49</td>
</tr>
<tr>
<td>2 columns</td>
<td>132</td>
<td>287</td>
</tr>
<tr>
<td>3 columns</td>
<td>870</td>
<td>1889</td>
</tr>
<tr>
<td>4 columns</td>
<td>5910</td>
<td>14393</td>
</tr>
</tbody>
</table>
IMPORTANT FINDINGS

• Different weaknesses surface in different scenarios
• Minimization Goal affects performances, especially at small / large storage constraints
• Solution Granularity depends on Workload, Approach, Budget
• Costing takes up majority of runtime for most approaches
IMPORTANT FINDINGS

• No overall “best” index selection algorithm
  • Workload
  • Storage Budget
  • DBMS
  • Runtime constraints
TODAY’S AGENDA

Overview
Index Selection Algorithms
Methods
Results
Parting Thoughts
PARTING THOUGHTS

Main Contribution

- A platform for evaluating index selection algorithms that abstracts away the cost model and DBMS*.

* If DBMS exposes interface for hypothetical indexes

Evaluate algorithms on equal footing

- Different dimensions needed to be more comprehensive
  - More workloads (Transactional)
  - Cost models which account for index maintenance

- “Fairness” in evaluation
  - Benefit vs Benefit/Storage – Apples to Oranges?
  - Workloads
REFERENCES
