Special Topics: Self-Driving Database Management Systems

Index Recommendation I

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

Overview of AutoAdmin
Architecture Deep Dive
Summary Analysis
Reflection
## PHYSICAL DESIGN

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>animal</th>
<th>...</th>
<th>net_worth</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Terrier</td>
<td>Dog</td>
<td></td>
<td>$800</td>
</tr>
<tr>
<td>1</td>
<td>Scout</td>
<td>Dog</td>
<td></td>
<td>$800</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>N</td>
<td>Kapi</td>
<td>Capybara</td>
<td></td>
<td>$1000</td>
</tr>
</tbody>
</table>

**SELECT** * FROM animals WHERE net_worth = $1000 **LIMIT** 1 -> O(N) to O(1)

**SELECT** * FROM animals WHERE animal = dog -> O(N) to O(LOG N)

**SELECT** name FROM animals **ORDER BY** name, animal **LIMIT** 100 -> O(N LOG N) to O(1)
PRIOR WORK

State of the Art in 1997:

→ Stonebraker suggested views could simulate databases\(^0\)

→ Index evaluation had a few approaches:
  ▪ Approaches based on query semantics and table statistics\(^1\)
  ▪ Expert systems using rules for decision-making\(^2\)
  ▪ Using internal components to compare different designs\(^3\)
The first version of AutoAdmin was the precursor to modern autonomous database tuning tools.

→ Implemented a hypothetical action framework that considered system overhead

→ Implemented a tuner that interfaced directly with DBMS internals

→ First tuner in DBMS industry
HYPOTHETICAL CONFIGURATION SIMULATION

1. **How can we estimate costs without executing queries?**
   The optimizer is the actual component of the system which chooses query execution. We can rely on its internal costing mechanism.

2. **How do we create a hypothetical configuration?**
   The simplest way would be to literally change the configuration to the hypothetical. This is costly/inefficient.
   What we really need is what the optimizer will use from the index. Densities and histograms.
SQL Server allows optimization of queries without executing them.

Using this mode and retrieving the query plan allows the system to determine the cost of a query.

Using the optimizer in this fashion ensures costing is consistent with actual execution.
Creating Hypothetical Indices

In order to generate the index, the system samples pages of the dataset until convergence (or a specified fraction) and creates density estimates and histograms.

5% sampling has < 4% error on query cost.

Other details:
→ Can’t put indexes in catalog
→ Handle in connection-specific manner
→ Allow rescaling
HYPOTHETICAL CONFIGURATION SIMULATION

Figure 2. Interfaces between HCA Engine and SQL Server

Hypothesised Configuration Analysis Engine
- Create hypothetical index
- Define configuration (HC mode)
- Optimize query "no-exec"

STORAGE ENGINE
- System Catalogs
- Index Statistics

OPTIMIZER

SQL SERVER 7.0
Autoadmin APIs

→ Hypothetical Configuration Simulation

**DEFINE WORKLOAD** `<workload-name>`
  [FROM `<file>` | AS `(Q_1,f_1),...,(Q_n,f_n)`]

**DEFINE CONFIGURATION** `<configuration-name>`
  AS `(Table_1, column_list_1),...(Table_i, column_list_i)`

**SET DATABASE SIZE OF** `<configuration-name>`
  AS `(Table_1, row_count_1),...,(Table_i, row_count_i)`

**ESTIMATE CONFIGURATION OF** `<workload-name>`
  FOR `<configuration-name>`

**REMOVE** [**WORKLOAD** `<workload_name>` | **CONFIGURATION** `<configuration_name>` | **COST-USAGE** `<configuration_name>`]
SUMMARY ANALYSIS

Instead of exposing the analysis data as part of the system catalog, they try to encapsulate what analysis might look like. They arrive at the following categories:

→ Workload Analysis
→ Configuration Analysis
→ Cost and Index Usage Analysis
SUMMARY ANALYSIS API

→ Summary Analysis

ANALYZE [WORKLOAD | CONFIGURATION | COST-USAGE] WITH <parameter-list>
[TOP <number> | SUMMARIZE USING <aggregation-function>]
BY <measure>
WHERE <filter-expression>
[PARTITION BY <partition-parameter>
IN <number> STEPS]
AutoAdmin was introduced to SQL Server with an index selection tool, introduced first in VLDB 1997. This paper improved:

→ Pruning the input candidate index set based on the configuration for individual queries in a workload

→ Enumerating configurations through a “seed” micro exhaustive enumeration, followed by a greedy search for improvements

→ A multi-column index search framework that iteratively grows the search space
A decade after this paper was released, it won VLDB 10-year Best Paper.

SQL Server continued to improve the tool.
→ Added new physical design structures (materialized views and partitioning)
→ Replacement of multi-column index generation procedure and introducing index merging
→ Multi-dimensional self-tuning histograms
→ Adding an alerting mechanism and online tuning
→ (Many more)
10 YEAR REFLECTION

The authors of this paper also share a reflection. The ten years since their original publication inspired a lot of work across the database industry and academia. That said, they acknowledge that achieving true self-tuning will still be a long road.

Nevertheless, they are optimistic on:

→ Defining physical design structures in a more lightweight fashion
→ Looking into ways to compare automated solutions in a meaningful way
→ Applying machine learning for self-tuning tasks
→ Innovative ground-up database architectures that manage complexity in favor of configurability
PARTING THOUGHTS

This tool is by no means “autonomous,” it falls under the “assistant” classification of automatic tools.

Huge factors aren’t considered – the cost of creating an index is large and the statistics collected are tied to a workload.

Capabilities of this tool are limited at best.
PARTING THOUGHTS

Rome wasn’t built in a day.

What does this paper provide?
→ What-if -> HypoPG
→ Implementing *inside* the database instead of using an external tool
NEXT CLASS

Index Recommendation II
REFERENCES