助力业务快速扩展
聊聊 TiDB 查询优化器的设计思考

董宇 @ TiDB 社区北京站

May 18, 2024

Optimizer is very hard ("harder than rocket science")

[Video] SQL Query Optimization: Why Is It So Hard to Get Right?

November 26, 2020 · Brent Ozar · Videos · No Comments

The first paper on cost-based query optimization was published in 1979 by Patricia Selinger from IBM Rese This paper laid the basic framework for optimizing relational queries that is still in place today. While there been many technical enhancements since 1979, query optimizers still fail to pick the best plan when faced complex SQL query.

In this talk I will describe the basic mechanisms used by modern query optimizers including plan enumeration, the use of histograms to estimate selective factors, and plan costing. I will also talk about a new approach to query optimization that I believe will revolutionize the optimization of queries in the cloud.

About the presenter – Dr. David DeWitt has positively wowed audiences at the PASS Summit over the years, consistently delivering amazing technical keynote presentations. You can read his bio at LinkedIn, or check out his Wikipedia page, or his past work at the University of Wisconsin. He could talk about pretty much anything, and I'd listen.

What is the role of the query optimizer?*

SQL does this part

The "query optimizer" does this part

*"Tell me **what** you want, not **how** to find it."*

Ted Codd (Inventor of RDBMS)

* "Planner" is another common term - prevalent in Postgres

Optimizer challenges from 10,000 ft view

- Problem to solve: **NP-hard**
- Problem nature: **“non-deterministic”** problem
    - Stats: “garbage in garbage out”
    - Cost estimation: Guy Lohman’s 25+10 years sigh (lament)
    - Enumeration: search space
- Strong assumptions vs. cruel reality


ACM SIGMOD Blog
Special Interest Group on Management Of Data

Blog Blog Archives About this Blog

Is Query Optimization a “Solved” Problem?

April 10, 2014 Guy Lohman 8 comments

Is Query Optimization a “solved” problem? If not, are we attacking the “right” problems? How should we identify the “right” problems to solve?

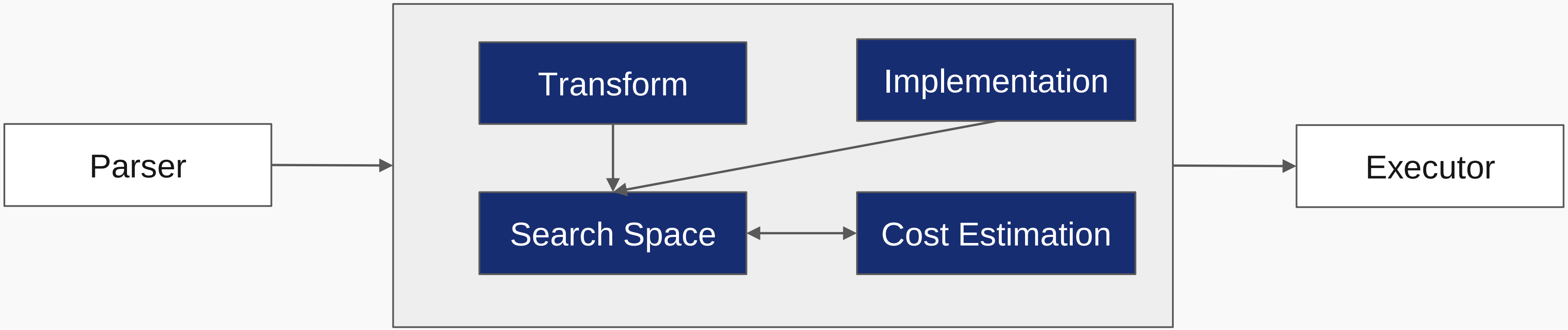
I asked these same questions almost exactly 25 years ago, in an extended abstract for a Workshop on Database Query Optimization that was organized by the then-Professor Goetz Graefe at the Oregon Graduate Center [Grae 89a]. Remarkably and quite regrettably, most of the issues and critical unsolved problems I identified in that brief rant remain true today. Researchers continue to attack the wrong problems, IMHO: ***they attack the ones that they can***, i.e., that they have ideas for, rather than the ones that ***they should***, i.e., that are critical to successfully modeling the true cost of plans and choosing a good one. Perhaps more importantly, that will avoid choosing a disastrous plan! At the risk of repeating myself, I’d like to re-visit these issues, because I’m disappointed that few in the research community have taken up my earlier challenge.

The root of all evil, the Achilles Heel of query optimization, is the estimation of the size of intermediate results, known as cardinalities. Everything in cost estimation depends upon how many rows will be processed, so the entire cost model is predicated upon the cardinality model. In my experience, the cost model may introduce errors of at most 30% for a given cardinality, but the cardinality model can quite easily introduce errors of ***many orders of magnitude***! I’ll give a real-world example in a moment. With such errors, the wonder isn’t “Why did the optimizer pick a bad plan?” Rather, the wonder is “Why would the optimizer ever pick a decent plan?”

A simplified optimizer on textbook

The nature of query optimization – search problem: *NP-hard* (really hard!)

- **Search space (plan alternatives) and search algorithms**
- **Enumeration (Transform/Implementation)**
- **Cost Estimation**



Let's Put the Optimizer Steps into Plain Text

Optimizer steps (significantly over-simplified)*:

1. **Rewrite/transform** the query to (attempt to) improve **predicate filtering**
2. Consider all viable **table** (and subquery) **join sequences**.
3. Map each (logical) plan step to **physical executable step**
4. Evaluate the number of **qualified rows** and **cost** for each **plan step**.
5. **Accumulate** the costs - and choose the **lowest cost plan**!

*Steps may be combined and/or be executed recursively - example: Predicate rewrites may generate the opportunity for other rewrites/optimizations as predicates are generated/simplified/pushed down. Rewrites may be rule or cost based (or both).

Optimizer challenges/risks of early steps*

1. Query rewrite

User: Can often rewrite. Lots of documented hints/tips.

DBMS: Independent of DBMS. Well documented.

2. Logical planning

DBMS: Very dependent on framework (System R, Cascades etc)

3. Logical to physical

User: Can control design options - eg. indexing

DBMS: Very dependent on implementation.

*Above are well understood for implementation - thanks to ongoing academic research and legacy database implementations (40+ years)

What about aater steps?

4. **Qualified rows** and **cost**.

DBMS: This can be HARD

User: Can influence statistics collection (input to optimizer)

5. **Accumulate** costs - choose **lowest cost** plan!

DBMS: Easy - it's math.
100 + 50 = 150

- #4 is where DBMS optimizer's struggle and rely too much on HINTS
    - HINTS are a valid solution provided you understand why(?) the hint is needed

Example from "another" DB's optimizer doc....

"Another"DB uses **PostgreSQL's cost-based optimizer**, which estimates the costs of each possible execution plan for an SQL statement. The execution plan with the lowest cost is executed. The planner does its best to select the best execution plan, but is **not perfect**.

Additionally, **the version of the planner used by "Another"DB is sub-optimal.** For instance, the cost-based optimizer is naive and assumes row counts for all tables to be 1000. **Row counts however play a crucial role in calculating the cost estimates. To overcome these limitations, you can use pg_hint_plan."**

What's hard about "qualified rows and cost"?

- Now we're getting to "**Cost Based Optimizer**" part of optimization
  - For each **operation** in the executable plan

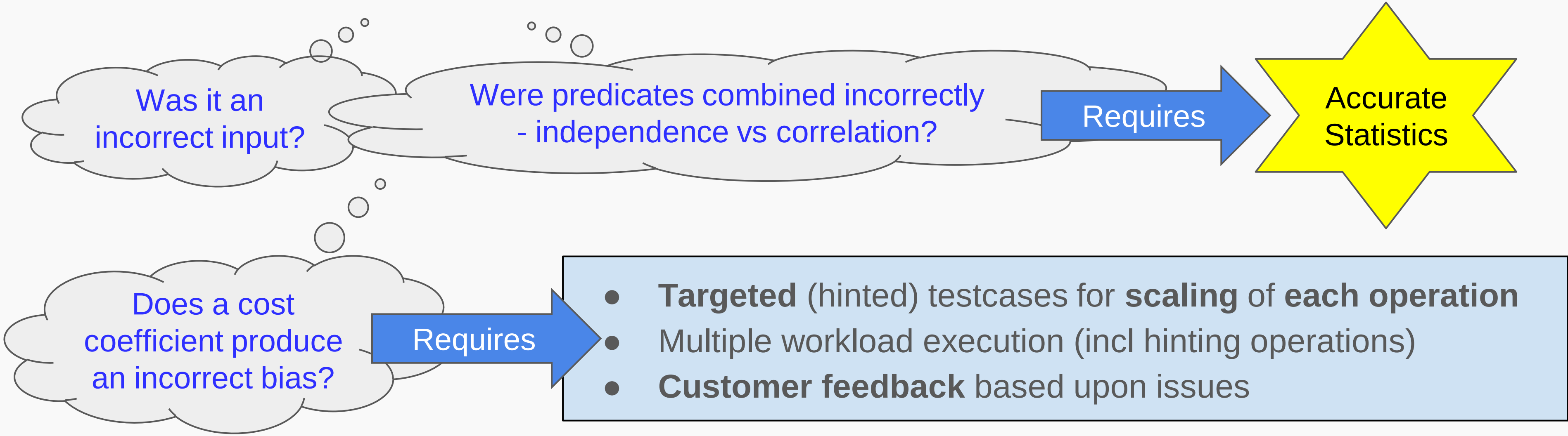
What is the operation's cost?

How many times do we perform this operation?

(# of rows * op#1cost) + (# of rows * op#2cost) = total cost

- "What's hard?" is
  - if the "total cost" is wrong - then one (or more) of the inputs was wrong

If an incorrect "cost" resulted in a poor plan?



- Cost model calibration is difficult for open source & individual contributors
  - Requires performance team, optimizer developer and execution developer
  - Ongoing "cost" adjustments are influenced by
    - Improvements to execution behavior of individual operations

Are there solutions?

- The optimizer's success rate can be improved by:
  - Providing "clarity" in choices
    - (relatively) Current/accurate statistics
    - Indexes targeted at the workload - most filtering predicates
    - Efficient execution choices where targeted indexes are not viable
  - Low risk (safe) estimates when statistics are stale or missing
  - Reduce frequency of plan selection*
    - Plan caching
    - Plan management (persisting plans outside of cache)
  - Hints for specific queries
  - Feedback loop to optimizer developers - to adjust balance/assumptions

Summary of challenges to overcome

Open challenges faced by traditional optimizer and TiDB optimizer

- Never sufficient and accurate STATs, or impossible to collect
- Optimization cost/overhead itself
- Diversified and evolving system architecture (computing & storages)
- More complex queries and applications
- More dynamic execution environment (e.g., cloud elasticity)
- More optimization objectives (e.g., cost/money, SLA, etc.)
- How to systematically evaluate an optimizer
- New and heterogeneous hardware (NVM/SCM, GPU/FPGA, RDMA, etc.)
- … …

TiDB Optimizer: from OLTP to HTAP

Pains and Gains when getting to HTAP:

- **More complicated optimization**: enlarged search space, complex AP oriented optimizations
- Subtle costing to differentiate plan choices b/t TiKV and TiFlash
- Optimization overhead
- Mitigate impact on mission critical OLTP workload
- … …
- **Leverage power of both engines** and *pushdown computation*;
- Natural **REAL-TIME analytical** workload processing;

TiDB Optimizer: HTAP optimization

Major HTAP related optimization:

- Two storage & coprocessor engines: TiKV/copr, TiFlash/MPP
- Access path selection of TiKV (index) and/or TiFlash
- Pushdowns: Agg, TopN, filter, join (TiFlash MPP), etc.
- MPP related planning (plan fragment, shuffle, etc.)
- Fine-tuned cost model

Additional Demands of Today's Applications

Larger data volumes - need to scale

Faster application deployment

Volatile/dramatic changes in data volumes

Moving away from legacy or vendor lock in

Embracing open source

Common RDBMS Features for Scale

- The following are common features to address the previously listed challenges
  - Automated statistics collection (auto-ANALYZE)
  - Query caching
  - SQL Plan management
    - TiDB differentiator - SQL Binding*
  - Sharding/partitioning
    - TiDB automatically shards by data range
    - Partitioning is optional (and complementary to sharding)
  - Index advisor/analysis to assist with targeted index design
    - TiDB differentiator - TiFlash*

TiDB Optimizer - Main Strengths/Differentiators

- Two main differentiators (to focus on)
  - TiDB's "distribution aware" Optimizer
    - Including HTAP/MPP Engine
  - Cross Database Binding
    - Allows reliable optimizer behavior with new (Saas) application rollout

- Two (other) important benefits from TiDB Optimizer
  - Auto-ANALYZE ← NOTE: Many RDBMSs remove the burden from users
  - Acknowledgement in estimation of missing/stale statistics
    - Examples include implied correlation, out-of-range

Demands of Today's Applications - Recap

- Compared to the application focus of legacy DBMSs……today sees:
  - Larger data volumes and need to (massively) scale
    - Distributed SQL databases
  - More volatile/dramatic change in data volumes
    - Data is moving more rapidly
  - Faster application deployment
    - Including SaaS "cloning" of new schemas/tenants
  - Customers moving away from higher cost legacy database implementations
    - Reduced (or desire to hire) skilled resources to (micro-) manage systems

TiDB - TiKV/TiFlash

Auto-Analyze, "smart" defaults/assumptions

Cross DB Binding

Who can you trust to provide the same level of performance?

TiDB Optimizer: lessons and thoughts

Some lessons and thoughts:

- Optimizer vs. better query engine (former usually gains most, CBO critical for join ordering)
- Cost model vs. cardinality estimate (latter matters more)
- STATS accuracy vs. collection impact on system stability
- Be cautious to introduce new stats (cm-sketch, feedback)
- Cardinality estimate often goes off (worthy improving)
- More indexes, more complexity (subtle costs in many cases)
- Left deep tree vs. Bushy tree join order (former solves 80% of cases)
- System R vs. Cascades (story behind Redshift and GP)
- Good plan vs. optimal plan (80% customers won't tell)
- Stable comes first before performance (avoid disaster plan)
- Earlier changes the better (avoid massive customer regression)
- Regression test (ideal: automated, diagnosis enabled)
- Advisors vs. Auto tuning (customers usually ignore advices)

Takeaways:

- Optimizer is very hard ("harder than rocket science")
  - HTAP optimizer is ***even harder***

- TiDB optimizer:
  - Keep evolving
  - Smart to help agile application development

[Video] SQL Query Optimization: Why Is It So Hard to Get Right?

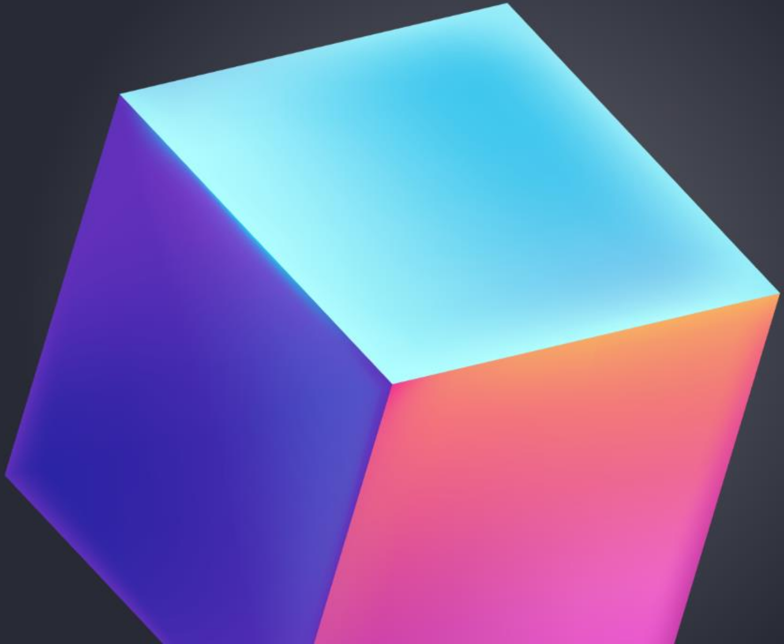
November 26, 2020 Brent Ozar Videos No Comments

The first paper on cost-based query optimization was published in 1979 by Patricia Selinger from IBM Rese
This paper laid the basic framework for optimizing relational queries that is still in place today. While there
been many technical enhancements since 1979, query optimizers still fail to pick the best plan when faced
complex SQL query.

In this talk I will describe the basic mechanisms used by modern query optimizers including plan enumeration, the use of histograms to estimate selective factors, and plan costing. I will also talk about a new approach to query optimization that I believe will revolutionize the optimization of queries in the cloud.

About the presenter – Dr. David DeWitt has positively wowed audiences at the PASS Summit over the years, consistently delivering amazing technical keynote presentations. You can read his bio at LinkedIn, or check out his Wikipedia page, or his past work at the University of Wisconsin. He could talk about pretty much anything, and I'd listen.

THANK YOU.



TiDB Optimizer: tuning practice

Common reasons caused bad plan/performance:

- Never-perfect optimizer (“**harder than rocket science**” – David DeWitt)
- Insufficient & inaccurate STATs vs. complex data distribution (skew, correlation)
- badly-written queries/applications
- bad DB physical design (partitions, indexes, etc.)
- code bugs
- etc.

Common DBA-ways to impact optimizer:

- manipulate stats or collect new stats
- rewrite query
- system parameters/knobs (game changing on cloud)
- introducing DB design changes (e.g., new indexes, re-distribute data, etc.)
- HINTs (*mixed love-hate*)

Tuning tools: *important*

TiDB Optimizer: query tuning

Some tuning tools & practice:

- SlowQuery/TopSQL
- Dashboard
- Visual Plan
- Optimizer hints/SPM
- Plan Replayer
- Manual rewrite
- … …