Cost-based Optimizer for Tajo

Proposal

Design Principles

- Extensibility
    - This optimizer will provide the interfaces for join enumeration algorithms and rewrite rules.
- Cost-based optimization
    - In order to find the best plan, the optimizer will be based on some cost model.

Details

Overall Phase

The query optimization will be performed as the following sequence:

1. SQL
2. Abstract Syntex Tree (AST)
3. Relational Algebra
4. Annotated Logical Plan
5. Apply some rewrite rules
6. Find the best join order
7. Apply other rewrite rules

Join Enumeration Algorithm Interface

- It will be an interface or an abstract class to take a list of tables, a join tree, or a mixed of them. It will result in the best-ordered join tree.
- This interface also takes a set of conjunctive normal form (CNF) predicates transformed from search condition (i.e., where condition).
    - With the CNF predicates, a join enumeration algorithm can immediately push down some predicates to joins and tables.
    - With the CNF predicates, a join enumeration algorithm can avoid cross joins.
        - It will reduce the search space of join enumerations.
- This interface should be able to access Tajo catalog in order to obtain statistics information of joined tables.
- In some case, a user wants to determine explicitly some parts of the join orders. The interface should consider this point.

Rewrite Engine

- I have a plan to implement a rewrite rule engine to support starburst style rewrite rules [1] and the rules mentioned in the paper [2].
- A rewrite rule will be also an interface or an abstract class to take a logical plan and result in the rewrote logical plan.
    - With this interface, we will be able to implement various rewrite rules.
    - The rewrite rule interface will have two primitive methods as follows:
        - boolean isEligible(LogicalPlan) - it checks whether this rule can be applied to this plan or not.
        - LogicalPlan Apply(LogicalPlan) - It returns a rewritten rule.
    - Some rewrite rules only focus on predicates. We can consider this observation for better design of the rewrite rule interface.
- With such rewrite rules, Tajo will have many benefits. I'm expecting that the following optimizations:
    - Redundant join elimination
    - Distinct elimination

- Predicate/projection pushdown
- Decorrelation of nested queries
- Predicate translation (e.g., IN <-> OR, IN -> JOIN, ...)
- Reuse of aggregation expressions
- Reordering predicates - more selective predicates will be evaluated earlier.
- Transformation of scalar subqueries
    - ALL and ANY indicators can lead to the max and min aggregation subqueries respectively.
- Other various rewrite rules

Utils

I also plan to implement the following common utilities to help us develop join enumeration algorithms and various rewrite rules.

- A data structure of the query graph model (QGM) that represents the relationships of query blocks and columns correlated among multiple query blocks.
- A join graph that provides a way to access tables and join conditions in a graph manner.
    - For example, each vertex indicates a table, and each edge indicates a join condition.
- Some algebraic utils for manipulating predicates.
- A visualizer util that generates figures of join graphs or query plans.

[1] Hamid Pirahesh et al., Extensible/Rule Based Query Rewrite Optimization in Starburst, ACM SIGMOD, 1992.

[2] Peter Gassner et al., Query Optimization in the IBM DB2 Family, IEEE Builtin, IEEE Data Eng. Bull, 1993.