The Volcano Optimizer Generator:
Extensibility and Efficient Search

Original Slides by:
Presentation: Alfred Pang
Discussion: Kati Radkehah
Modified by:
Rachel Pottinger
Nima Hazar

Outline
- Introduction
- Main concepts of the Volcano optimizer generator
- Search strategy
- Comparison with EXODUS
- Other researches
- Conclusions

The Generator Paradigm

This is not the first time for this approach (EXODUS).
Volcano improves on the work of EXODUS: ease of use, expressiveness.

Volcano Requirements
- Efficient & extensible for query and request processing for object-oriented and scientific database systems
- Required improvements
  - Stand-alone tool
  - More efficient search (time and memory)
  - Extensible support for physical properties (sort)
  - Permit use of heuristic and data model semantics (directed search, cost functions)
- Higher performance
  - Allowing query optimization to be more tuned towards the application
  - (Expert) User optimize

Design Principles

Model Specification

Optimizer Generator

Optimised Source Code

Compiler and Linker
Design Principles

- Fundamental design decisions
  - Relational algebra (logical and physical)
  - Specify knowledge by independent rules – Ensures modularity
  - Map queries to same algebraic equiv as Volcano’s input – Others use multiple intermediate levels
  - Rule compilation rather than interpretation – Query optimization is CPU intensive and time consuming
  - Search engine is base on dynamic programming

Optimizer: Input/Output

- Input: User Query => Logical algebra expression
- Output: Algorithms to access physical storage => Physical algebra expression
- Optimizer – mapping a logical algebra expression into the optimal physical algebra expression

Volcano: Input/Output

- Optimizer implementor provides
  - Set of logical operators
  - Algebraic transformation rules (logical -> physical)
  - Set of algorithms and enforcers
  - Implementation rules (operators to algorithms)
  - Cost functions for basic arithmetic and comparison
  - Applicability function for each algorithm and enforcer
  - Property function for each operator, algorithm and enforcer
- Output: Generated optimizer
  - Seem to be a lot of code

Volcano Plan Search Engine

- For choosing among the many possible plans
- Search engine is same for all generated optimizers, linked automatically with the generated code
- Use backward chaining – It explores those subqueries and plans that participate in the larger expression.
- Find costs of promising moves (transform, algorithm, or enforcer)

Volcano Plan Search Engine

Discussion Question 1 - Cost

The authors state that they're allowing the cost function to be changed to whatever the user wants to model. They give as examples estimated elapsed time, estimated CPU time, or I/O time. Since System R, people have largely estimated based on I/O.

What kinds of other things can you imagine people using?
Comparison With EXODUS

- Volcano solved EXODUS problems
  - Modifications required for unforeseen algebras
  - MESH data structure – time and space complexities
  - Reanalyzing overhead of random transformation in MESH
- Functionality and Extensibility
  - EXODUS did not consider logical expressions together with physical properties in optimization cost. (Volcano does)
  - Ability to specify required physical properties – powerful and extensible
  - Volcano algorithm is top-down (lower levels are explored only when warranted). EXODUS does the transformations whether they were part of the currently most promising.

How good was it?

Discussion Question 2 – Experiment

This paper uses a small data model and 50 queries at each complexity level as the experimental data.

Do you think their experimental evaluation is sufficient? Is it possible that these experimental data are biased?

Comparison to Starburst

- The design of the cost-based optimizer is focused on step-wise expansion of join expressions
- Starburst has a hierarchy of intermediate levels; harder to see interactions. Volcano uses an algebraic approach which paper claims to be easier to understand.
  - Which level of the hierarchy is the right place for a multi-way join algorithm?

Comparison With EXODUS

- Volcano took less time to optimize.
- EXODUS optimizer generator measurements were quite volatile and took a lot of memory.
- EXODUS’s generated optimizer and search engine do not explore and exploit physical properties and interesting orderings.

Comparison to Starburst

- New operators are integrated at the query rewrite level – optimization in this level is heuristic
- Query rewrites in Starburst does not include cost estimates.
- Although paper is critical of this, Volcano does allow for heuristic transformations to be specified.
Summary

- Remove the restrictions on the search strategy imposed by the earlier designs
  - EXODUS – cost analysis after a transformation
  - Starburst – one level heuristic optimization, the other level cost-sensitive exhaustive search
- Use physical properties to direct the search
- Subexpressions can be optimized multiple times – binary operations with multiple physical properties
- The internal structure for equivalence classes is modular and extensible to support alternative search strategies.
- Separation of logical and physical algebras makes specification and modification easy and search engine efficient

Discussion Question 3 - Optimizers

Starburst has a single but extensible optimizer, while Volcano supports the existence of multiple optimizers.

1. Why would we want several optimizers? Is it OO specific, or is it more general?
2. In terms of extensibility, performance and ease-of-use which model is better and in which circumstances Volcano or Starburst is favored?