Optimizing Queries across Diverse Data Sources

Laura M. Hass, Donald Kossmann,
Edward L. Wimmers, Jun Yang

IBM Almaden Research Center, 1997

Presentation by Kevin Quan (kquan@uwaterloo.ca)

Overview

- Problem Definition
- Architectural view of Garlic
- Query Plan Generation
- Query Optimization
- Conclusions & Discussion

What are they trying to solve?

- Use a middleware system for heterogeneous sources
- Optimize queries over sources with varying query processing capabilities
- Use a cost-based model for optimizing
- Implementation of approach in Garlic

What are their contributions?

- Gain the benefits of optimizations in each source
- Extensible by the use of wrappers which are:
  - Independent
  - Evolvable

What is Garlic?

- Global-As-View
- Wrappers act as the interface between query services and data sources
- Query services contain query language processor and distributed query execution engine
- Catalog contains global/local schemas
- Query language processor generates execution plan based on input
- Query execution engine passes sub-queries to wrappers and assembles final result.
- Assembly may include performing joins, applying predicates, sorting, aggregates

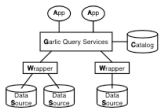

Figure 1: Garlic System Architecture

What do wrappers do?

- Wrappers can wrap various types of data sources
- Garlic wrappers are specific to Garlic -- provides interface to data source using Garlic's internal protocols
- Data described in an OO model, methods can be applied on data.
- Data source notifies wrapper of capabilities using rules
- Wrapper does not have to reflect full query functionality of data source

What are STARs?

- STARs = STrategy Alternative Rules
- Rules are high-level, declarative, compact specification of legal alternatives
- STARs define high-level constructs from low-level database operators or other STARs

$$\text{JoinRoot}(T1, T2, P) = \begin{bmatrix} \text{PermutedJoin}(T1, T2, P) \\ \text{PermutedJoin}(T2, T1, P) \end{bmatrix}$$

- Reference: G. Lohman. **Grammar-like functional rules for representing query optimization alternatives**. In *ACM SIGMOD Conf.*, Chicago, 1988.

How are plans constructed?

- Tuples are operated upon by POPs (Plan Operators)
- A POP generally corresponds to one executable operator
- POPs include: join, sort, filter, fetch, temp, scan, pushdown (work to be performed by source)
- POPs have properties that describes the specifics of the operations.
- *Source* property records where output stream comes from (needed?)

An Example of a Plan

- PushDown POP performs operations on the data source
- Data sources only return OID
- Wrappers take PushDowns and performs them on sources by translation into query or API calls
- Source property shows where execution occurs
- Properties of POPs are functions of parent POP (I.e. predicates)
- Additional properties: cost, card(inality)

Figure 3: One Possible Query Plan for:
`SELECT m.Body FROM Inbox m,Classes c WHERE m.Subject=c.Course AND c.Prof='Aho'`

What do STARs do?

- STARs determine how POPs and other STARs can be combined in a plan.
- General form:

```
STAR(params) ::== ∀ e ∈ set: f1(f2(…),f3(…),other args)
                  [if condition(args)]
```

- Example of a conditional:

```
FetchCols(T,C,Plan) ::== Fetch (T,C',Plan)
                    if C' ≠ ∅, C' = C-Plan.Columns
```

- Example of multiple plans:

```
DamStream({Plan}) ::== ∀ e ∈ {Plan}: Scan(Temp(p))
```

How is plan enumeration performed?

- `AccessRoot` STAR is used to create plans to select all attributes used in query (no real variability in plans, performs a PushDown).
- `JoinRoot` STAR is used to create plans to perform joins.
- `FinishRoot` STAR is used to include any missing parts of the query (i.e. projections, ordering)
- Pruning for query optimization performed throughout to minimize number of plans to enumerate.

How do they do Query Optimization?

- Cost of plans derived from:
  - Processing costs (estimated from cost model of CPU, I/O)
  - Communication costs (estimated using constants in catalog)
  - Cost to initiate subqueries & methods (estimated using constants in catalog)
  - Wrapper costs (estimated by wrapper)
- Plans are pruned upon enumeration
  - Plan A not used as building block for more complex plan if cheaper alternatives available
  - Plans with unique properties are not pruned

How are data source capabilities determined?

- Wrapper implements STARs that describe the capability of each data source.
- STARs follow POP structure mentioned previously
- Simple STARs can model *basic* capabilities of data sources
- Complex capability is arguably not needed as Garlic's query engine can make up for it.
- Wrapper can iteratively add STARs to:
  - Introduce source quickly into mediated schema
  - Improve performance

Conclusions

- One of first query optimizers using dynamic programming (STARs)
- Individual data sources are represented to Garlic by wrappers
- Garlic distributes query using PushDown operators
- No experimentation results
  - Experimentation on system with DB2, Oracle, ObjectStore, image processing system, Lotus Notes, & web sources.

What did I like?

- Wrapper STAR approach is great for quickly adding new source to integration system and incremental additions
- First look at a prototypical GAV system.

Discussion Topics

- Can Garlic handle complex PushDowns to data sources?
- Should query complexity be solely offloaded to Garlic?
- How do you quantify costs of a data source?
- How is global<->local schema translation handled?
- Is the object data model used by Garlic worthwhile/efficient?
- Can we attain the same characteristics for the wrappers without using STARs?