SMOKE: Fine-Grained Lineage Capture at Interactive Speed

Fotis Psallidas
totos.cs.columbia.edu

Eugene Wu
ewu@cs.columbia.edu

Introduction

Lineage at the core of many applications
Visualization, debugging, diagnostics, profiling, why-not, interpretability, ...

Overhead of lineage capture can cripple query engine performance
Smoke introduces principles to reduce overhead from >100x slowdown to ~1.3x slowdown

Related Approaches

Considerable redundant work

Lazy
- Rewrite lineage query into SQL
- backward_trace(σ_{id = x}, B) = \gamma_{id = x}(A)

Logical Denormalized
- Rewrite base query to propagate lineage as annotations

Logical Normalized
- Driver rewrites query to propagate lineage as normalized relations

Physical
- Send lineage data to separate subsystem

Lineage Capture

4 Design Principles

P1. Tight Integration
- Instrument operators to write lineage idxs

P2. Reuse work
- Lineage indexes ~ Hash tables
- Intra-plan hash table reuse

P3. Apriori Knowledge
- Don’t capture if not used

P4. Lineage Consumption
- Push computation into lineage capture

SMOKE Overview

Example: Group-By

Two Lineage Capture Paradigms

Defer
- Doesn’t block the query
- Defers capture after operator

Inject
- Faster overall but blocks query execution
- Generate results

Example

Crossfilter Performance

Profiling Performance

Capture Overhead (Group-By)

Lineage capture can be
- Fast enough to be interactive
- Competitive with hand-written implementations
- Useful for data-intensive interactive apps

Interested in More?

[VLDB18] Smoke: Fine-Grained Lineage At Interactive Speed
[SIGMOD18] A Deep breath of Data-Intensive Lineage Applications
[HILDA18] Provenance for Interactive Visualizations
[CIDR17] Combining Design and Performance in a DVMS