Arachne: Big Lineage Analysis for Graph Analytics

Vicky Papavasileiou[⊙], Wojciech Kazana[⊙], Alin Deutsch[⊙] and Kenneth Yocum[⊙+] *UC San Diego[⊙], Illumina, Inc.[+]*

Analyze lineage of large scale graph analytics

Challenge : Lineage attains big data scales

- PageRank on 1M vertices graph produces provenance graph 51M vertices and 300M edges

Contributions:

- Provenance Model: Formal semantic model for parallel graph processing frameworks
- Low-overhead lineage capture and storage framework
- Query Language: Declarative language for querying provenance

Optimize fixed-point analytics

- Machine Learning algorithms run iteratively until fixed-point is reached
- Capitalize on inexact computation by proposing fewer iterations

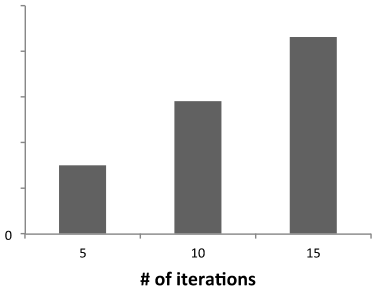
Optimization library

- Declarative queries that indicate approximate optimizations (αptimizations)
- Return **two** sets of vertices: first contains vertices whose execution can be skipped, second contains vertices that if skipped will distort final result

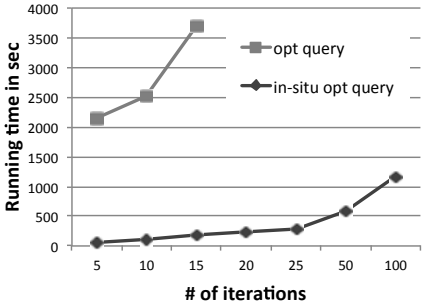
Experimental results

Alternating Least Squares collaborative filtering algorithm

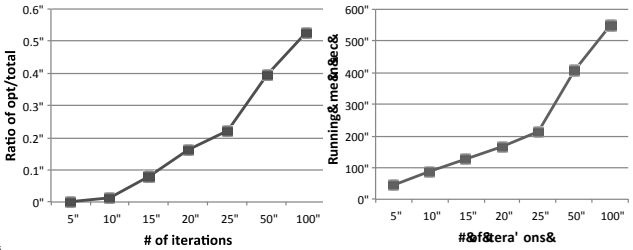
- Captured and analyzed large scale provenance graph
  - Loaded the provenance graph in GraphLab
  - Input graph: 97K vertices, 3.8M edges



- Runtime overhead
  - Querying the captured provenance graph x6-8.4 overhead
  - In-situ query evaluation only 5% overhead



- Optimization query results



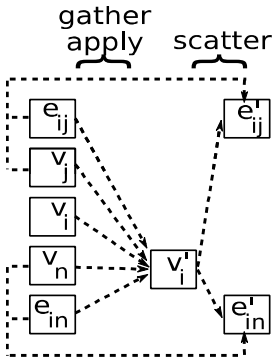
Provenance model

GraphLab GAS computation model

- Data resides on both edges and vertices
- All vertices evaluate same computation (vertex program)
  - Gather data from adjacent vertices and edges
  - Apply function to compute new value
  - Scatter new value to adjacent edges
- Computation progresses in iterations
  - A vertex executes if a neighbor signals it or if the maximun number of iterations has not been reached

Provenance graphs

- Add node to provenance graph for every edge and vertex, vertex iteration
- Add two types of edges: *input* edges and *evolution* edges



Query language

- Datalog with negation and with inflationary semantics
  - Declarative specification of "classical" lineage queries s.a tracing as well as αptimization queries
  - Allows efficient parallelization and incremental evaluation of queries
  - In-situ evaluation: Evaluate queries along with actual run of analytic ➔ No need to capture full provenance graph

Optimization library

- General idea: If all neighbors of vertex u skip signaling it, then u will not execute in next iteration
- Two conditions must hold:

1) No_signal: All neighbors must not signal
2) Safe: The impact of skipping this iteration is negligible

Optimization candidates: both conditions hold

Counter examples: second condition violated

- PageRank Example:
  - If all neighbors change less than ε (no_signal) and u changes less than e (safe) then u is an opt. candidate
  - If all neighbors change less than ε (no_signal) but u changes more (not safe) then u is a counter example