http://www.cs.umd.edu/~kostasx

GRADES 2017

GraphGen: *Adaptive* Graph Processing using Relational Databases

Konstantinos Xirogiannopoulos, Virinchi Srinivas, Amol Deshpande
University of Maryland, College Park

Graph + Relational Design

Graph Analysis Tasks *Vary Widely*

- Different types of Graph **Queries**
- *Continuous* Queries / ***Real-Time*** Analysis
- Batch Graph Analytics
- Machine Learning

Many different ways to deal with graph data

- Graph Databases (Neo4j, OrientDB, RDF stores)
- Distributed Batch Analysis Frameworks (Giraph, GraphX, GraphLab)
- In-Memory Systems(Ligra, Green-Marl, X-Stream)
- Many research prototypes/custom indexes

1. *Graph* Frontend, *Graph* Backend

- RDF, Property Graph databases
- Focus on ***graph queries***
- **Issue**: Requires a ***complete buy-in***

2. *Graph* Frontend, *Relational* Backend

- Some XML/RDF/ Property Graph Database (e.g., SQLGraph, Titan)
- Graph analytics frameworks (Vertexica, Grail)
- **Issue:** Requires complete buy-in; limited expressivity for analytics

3. *Relational* Frontend, *Relational* Backend, *Graph* Engine

- Using a ***graph engine*** to efficiently process **SQL** queries over *relational* data
- Mostly to tackle queries involving ***long series of joins.***

4. *Graph+Relational* Frontend, *Relational* Backend (*GraphGen*)

- Aster Graph, SAP Graph Engine
- ***Enterprises*** have existing relational databases with ***rigid schemas***
- Need to enable analysis of the ***hidden*** graphs within them
- While continuing to support SQL queries/analytics

GraphGen

GraphGenDL

- Declarative specification of **GraphViews** over the database
  - User specifies **Nodes** and **Edges**

CoAuthors GraphView

```
CREATE GRAPHVIEW CoAuthorsWeighted AS
    Nodes(ID, name) :- Author(ID, name).
    Edges(ID1, ID2, wt=$COUNT(pub)) :-
AuthorPub(ID1, pub),
 AuthorPub(ID2, pub).
```

Ego-Graphs Multi-GraphView

```
CREATE GRAPHVIEW AuthorEgoNetworks(X)
WHERE Author(X) AS
    Nodes(X, name) :- Author(X, name).
    Nodes(ID, name) :- AuthorPub(X,pub),
      AuthorPub(ID,pub), Author(ID, name).
    Edges(ID1, ID2) :- AuthorPub(ID1, pub),
      AuthorPub(ID2, pub).
```

GraphGenQL

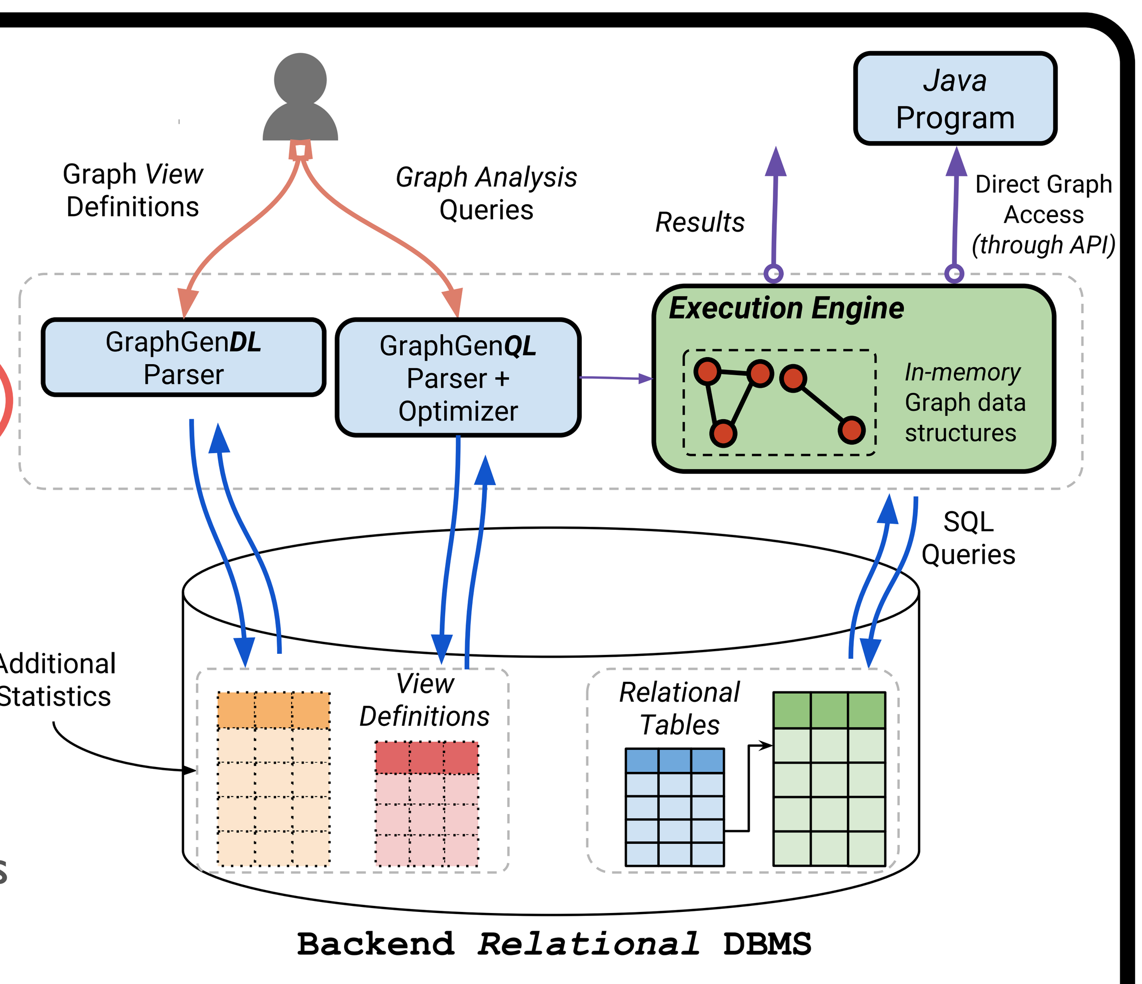
- Support for subgraph pattern matching query languages like SPARQL etc.
- ***Datalog*** is a natural fit!

Find all ***triangles*** where 'ML'>'DB'>'AL'

```
USING GRAPHVIEW CoAuthors
 Triangle(X, Y, Z) :- Nodes(X, _, "ML"
 ),Nodes(Y, _, "DB"),Nodes(Z, _,
 "AL" ),Edges(X, Y),Edges(Y,
 Z),Edges(X, Z).
```

Specifying Analysis Tasks

- ***Vertex-centric*** works for many situations but ***not very expressive***. We plan to support:
  - ***Direct access*** to graph views for complex programs
  - Datalog-based DSL (build upon languages like ***Socialite***)



Declarative language abstractions enable optimizations for adaptive processing

Opportunities and Challenges

Where to Execute Tasks?

- Dependent on workload, rate of updates, rate of queries…
- Usually ***in-memory*** is faster, but ***ETL*** may not be worth it
- Other issues: ***Large-output joins***, and ***selectivity estimation errors*** associated with them.

Key Challenge: Develop accurate ***cost models,*** workload monitoring ***tools***, and ***optimization*** techniques

- ***How much*** of the graph do we *pre-compute* / materialize?
- ***Incremental view maintenance*** for graph-views may prove ***challenging***

Query Rewriting

- Auto-generated SQL can consist of many blocks
- Many ways of writing equivalent SQL queries, optimization can be challenging
  - Could define edges using the **WITH** clause, or as a **VIEW**

Key Challenge: Optimizing SQL for graph analytics exposes ***gaps in query optimizers***

```
WITH Nodes as (...)
WITH Edges as (...)
(SQL for answering query)
```

DISTINCT

```
Create VIEW Nodes as (...)
Create VIEW Edges as (...)
(SQL for answering query)
```

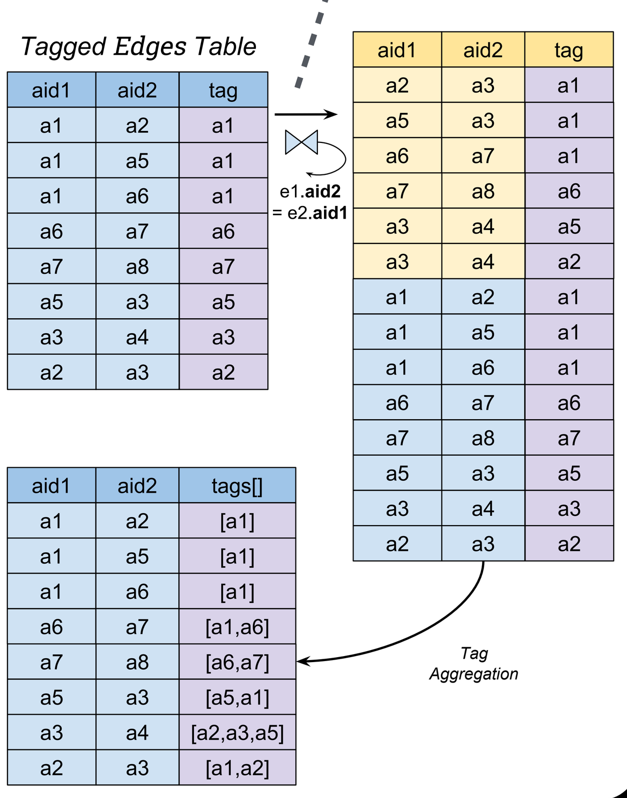
DISTINCT

Optimizing the Extraction of Multi-Graph Views

- Typically one would need to generate a ***separate query*** for each graph
- We employ a technique called ***result-tagging***

Key Challenge: Develop a systematic approach towards execution over ***collections of graphs***

Find edges 1-hop away from source (tag) & union with Tagged Edges Table



Tag-list: *which* ego-graphs **include** edge

Preliminary Experiments

Query	DBS1	DBS2	MySQL	PostgreSQL
With (at edges)	**1**	**1.62**	NA	**13.8**
With (at the end)	53.028	2.99	NA	37.8
View (at edges)	1.054	2.07	**3.01**	15.6
View (at the end)	51.92	77.13	538.19	35.991
On Base Table	46.45	74.878	678.87	36.160

Triangle Counting (small); time in *seconds*

Query	DBS1	MySql	PostgreSql
Using With (in edge)	48.05	NA	477.56
Using With (at the end)	2404	NA	1499.33
Using View (in edges)	44.72	357.59	811
Using View (at the end)	2377.61	>3600	1774.54
Directly On Base Table	2348	> 3600	1790

Triangle Counting (large); time in *seconds*

Dataset	Triangle Counting	Triangle Pattern	ETL
small	0.169	0.001	2.049
large	6.723	0.015	17.52

In-memory execution; time in *seconds*

Query	DBS1	DBS2	MySQL	PostgreSQL
With (at edges)	14.765		NA	0.749
With (at the end)	4.59		NA	**0.704**
View (at edges)	15.557		**4.26**	2.193
View (at the end)	**4.25**		20	11.063
On Base Table	8.612		22.69	3.089

Pattern Matching where area='ML' (large); time in *seconds*

- Query ***rewrites*** lead to significant differences in ***performance***
- ***In-memory*** execution is usually *faster* but ***when its warranted***

dataset	#rows	nodes	edges
small	100,000	1,639	55,436
large	500,000	15,741	529,434

Dataset sizes