Extending In-Memory Relational Database Engines with Native Graph Support

Mohamed S. Hassan[1], Tatiana Kuznetsova[1], Hyun Chai Jeong[1], Walid G. Aref[1], Mohammad Sadoghi[2]
[1]Purdue University, West Lafayette, IN [2]University of California, Davis, CA

Motivation

- Graphs are ubiquitous (e.g., road networks, social networks, biological networks, data-center networks)
- Specialized graph systems are not as mature as RDBMSs
- Graph-Relational queries are pervasive in many applications
  - Queries containing graph operations, (e.g., shortest-paths) and relational predicates
  - E.g., select specific users from relational tables, then find their nearest hospitals using shortest-path over a road-network
- Vanilla RDBMSs cannot evaluate deep-traversal queries efficiently
  - Large intermediate results of the join operations
  - Inaccurate cardinality estimation

Existing Approaches

- Native Relational-Core
  - Deep-traversal queries are inefficient to evaluate
  - Graphs are encoded in complex schemas
- Native Graph-Core
  - Graphs are extracted from RDBMS into graph-core
  - Graph updates require graph re-extraction
  - Queries cannot reference non-extracted relational data

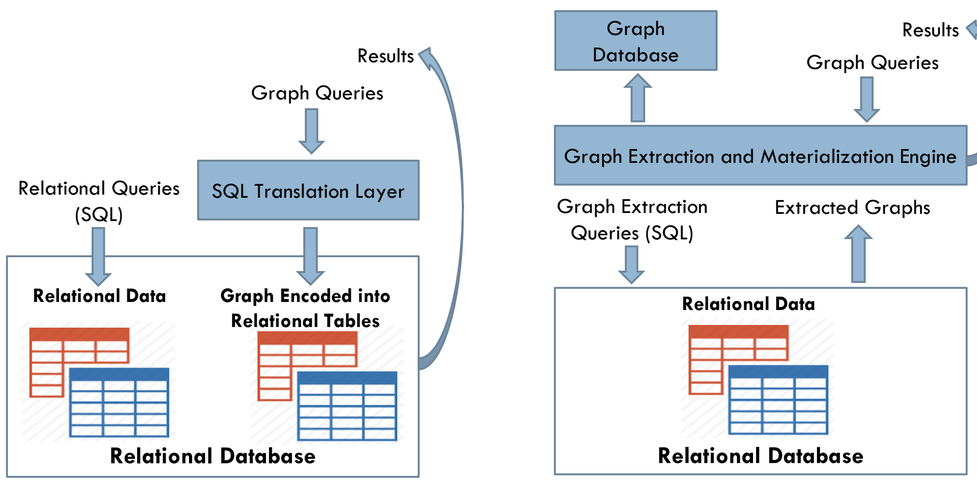

Figure 1: Existing approaches for leveraging relational databases to support graph processing.

Proposed Approach: Native G+R Core

- Represent graphs as native graph structures
- Extend SQL to reference graphs in queries
- Support cross-data-model QEPs
- **GRFusion** realizes the Native G+R approach

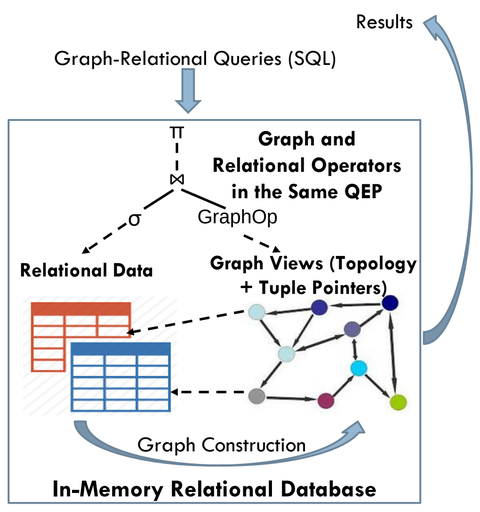

Figure 2: The Native G+R Core Approach.

Traversal Operators

GRFusion introduces the **PathScan** logical operator

- Operate over a graph view
- Has three corresponding physical operators: **DFScan**, **BFScan**, and **SPScan**
- Specify the vertexes to start the traversal from
- The output extends the standard relational tuple, hence, the output can be ingested by any relational operator

Creating Graph Views in GRFusion

```
CREATE UNDIRECTED GRAPH VIEW SocialNetwork
VERTEXES(ID = uId, lstName = lName,
    ↪ birthdate = dob)
FROM Users
EDGES (ID = relId, FROM = uId1, TO = uId2,
    ↪ startDate = sDate, relative =
    ↪ isRelative)
FROM Relationships
```

Figure 3: Creating a Social-Network Graph-View Example.

Lightweight Graph Views in GRFusion

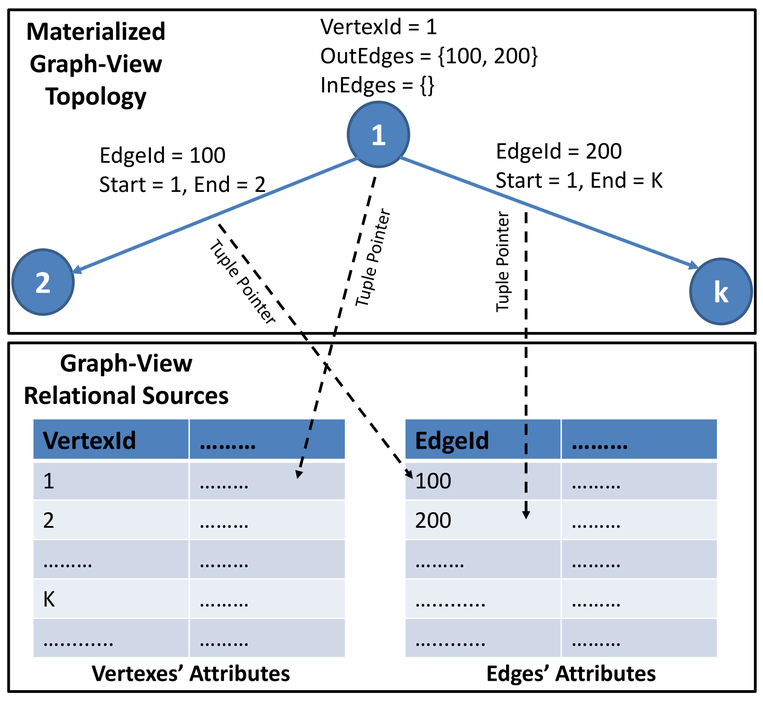

Figure 4: A graph view materializes the topology and holds pointers to the relational data of the vertexes and the edges.

The PATHS Construct and Cross-Model QEPs

```
SELECT PS.EndVertex.lstName
FROM Users U, SocialNetwork.Paths PS
WHERE U.Job = 'Lawyer' AND PS.StartVertex.Id
    ↪  = U.uId AND PS.Length = 2 AND PS.
    ↪ Edges[0..*].StartDate > '1/1/2000'
```

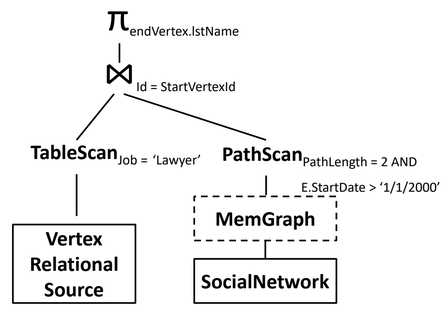

Figure 5: GRFusion joins a table with a graph-view traversal-operator.

Graph-Traversal Query Examples

```
SELECT PS.PathString
FROM Proteins Pr1, Proteins Pr2, BioNetwork.
    ↪ Paths PS
WHERE Pr1.Name = 'Protein X' AND Pr2.Name =
    ↪ 'Protein Y' AND PS.StartVertex.Id =
    ↪ Pr1.Id AND PS.EndVertex.Id = Pr2.Id
    ↪ AND PS.Edges[0..*].Type IN ('covalent
    ↪ ', 'stable')
LIMIT 1
```

```
SELECT TOP 2 PS
FROM RoadNetwork.Paths PS HINT(SHORTESTPATH(
    ↪ Distance)), RoadNetwork.Vertexes Src,
    ↪  RoadNetwork.Vertexes Dest
WHERE PS.StartVertex.Id = Src.Id AND PS.
    ↪ EndVertex.Id = Dest.Id AND Src.
    ↪ Address = "Address 1" AND Dest.
    ↪ Address = "Address 2"
```

Figure 6: Reachability and Shortest Path Queries in GRFusion.

Experimental Results

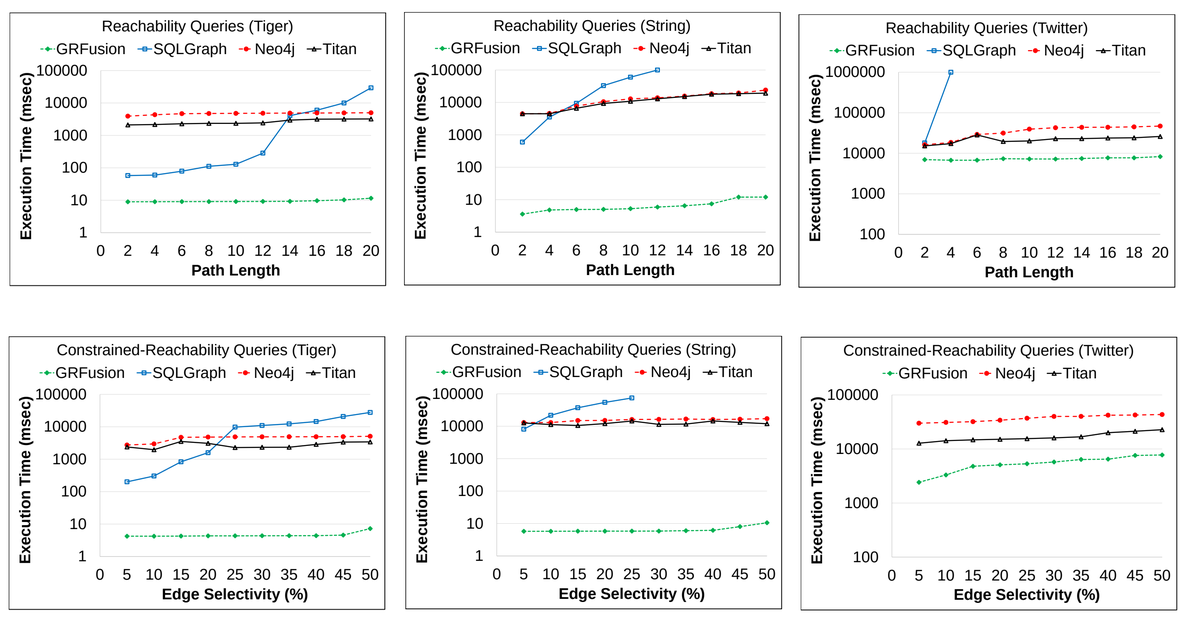

Figure 7: GRFusion achieves up to four orders-of-magnitude query-time speedup for constrained and unconstrained reachability queries.