GRainDB:
A Hybrid Graph-Relational DBMS
Semih Salihoğlu
Joint w/ Guodong Jin
Many Appeals of a Relational-core Hybrid System

- Hybrid System: An extended RDBMS with *graph modeling, querying,* and *visualization* capabilities.

1. No Perfect Data Model
   - **Tables**
     - Legacy data
     - Non-binary relations of entities
     - Good for normalization (e.g., zipcodes, days, dates)
   - **Graphs**
     - Arguably closer to developers’ mental model of real-world entities and relationships

2. No Perfect Query Language
   - **SQL**
     - Very popular and established
     - Suitable for standard data analytics, preparation, etl...
   - **Graph Query Languages**
     - Easier for recursive queries
     - MATCH a-[Transfer*]->b
     - WHERE a.owner=Alice

3. Cheaper and quicker than building a completely separate GDBMS
GRainDB Vision

**Neo4j Bloom-style Graph Visualization**

```
SELECT DISTINCT Address.zipcode
FROM (a:vPers)-[:eKnows*1..3]->(b:vPers),
     Address
WHERE a.name=Alice AND b.addID=Address.ID
```

**G-SQL-style Seamless Table/Graph Querying**

**Graph Modeling**

+ Pre-defined pointer-based joins
+ Factorization
+ Worst-case optimal joins
+ Recursive joins
Predefined Pointer-based Joins in GDBMSs

- Primary Difference in Join Processing in GDBMSs vs RDBMSs:
  - Pointer vs Value-based joins

MATCH a-[[:Trnsfr]]->b-[[:Trnsfr]]->c
WHERE b.owner = "Alice"

- Adjacency Lists = An Index Over Edges
- ID-based Nested Index Loop Joins
Predefined Pointer-based Joins in GRainDB

Step 1: Predefine a Primary Key-Foreign Key Join E.g.:

FROM: Accounts, Transfers
WHERE Accounts.owner = Transfers.From

Columnar RDBMS use Row IDs (RIDs) as system-level pointers

<table>
<thead>
<tr>
<th>Accounts</th>
<th>Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>owner</td>
</tr>
<tr>
<td>1</td>
<td>Alice</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
</tr>
<tr>
<td>3</td>
<td>Carol</td>
</tr>
<tr>
<td>4</td>
<td>Alice</td>
</tr>
<tr>
<td>5</td>
<td>Alice</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Step 1: RID Materialization and RID Index

SELECT a.owner, c.owner
FROM Acc a, b, c, Trn t1, t2
WHERE b.owner = Alice AND
a.owner=t1.From AND t1.To=b.owner AND
t1.To=t2.From AND t2.to=c.owner

 Accounts
 RID  owner
 1  Alice
 2  Bob
 3  Carol
 ...

 Transfers
 RID  F  RID  From    To     amount
 1   11 Alice  Bob    700
 2   22 Bob    Carol  800
 3   33 Carol  Alice  900
 4   14 Alice  Dan    500
 5   15 Alice  Liz    400
 ...

 RID Index

 Accounts
 RIDs  Transfer
 1     1 4 5
 2     2
 3     3

 RID Index

 Accounts
 RID  owner
 1  Alice
 2  Bob
 3  Carol
 ...

 Transfers
 RID  F  RID  From    To     amount
 1   11 Alice  Bob    700
 2   22 Bob    Carol  800
 3   33 Carol  Alice  900
 4   14 Alice  Dan    500
 5   15 Alice  Liz    400
 ...

 RID Index

 Accounts
 RIDs  Transfer
 1     1 4 5
 2     2
 3     3

 RID Index
Step 2: Rule-based Query Planning

```
SELECT a.owner, c.owner
FROM Acc a, b, c, Trn t1, t2
WHERE b.owner = Alice AND
  a.owner = t1.From AND t1.To = b.owner AND
  t1.To = t2.From AND t2.to = c.owner
```

1. Replace some HashJoins -> SIPJoin or SIPJoinIdx
2. Replace some Scans -> ScanSemiJoins (ScanSJ)
Step 2: Rule-based Query Planning

SELECT a.owner, c.owner
FROM Acc a, b, c, Trn t1, t2
WHERE b.owner = Alice AND
a.owner=t1.From AND t1.To=b.owner AND
  t1.To=t2.From AND t2.to=c.owner
Step 3: Sideways Information Passing & Semijoins

```
SELECT a.owner, c.owner
FROM Acc a, b, c, Trn t1, t2
WHERE b.owner = Alice AND
a.owner = t1.From AND t1.To = b.owner AND
t1.To = t2.From AND t2.to = c.owner
```

<table>
<thead>
<tr>
<th>Accounts</th>
<th>Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID</td>
<td>F(RID)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Use RIDs as pointers
- All scans are sequential unlike nested loop joins of GDBMSs
Experiment: LDBC Social Network Graph Benchmark

- LDBC 10 Benchmark: ~10GB
- Dual 2.6GHz Intel CPU, 256GB RAM
- In-Memory Performance
Guodong Jin

Making RDBMSs Efficient on Graph Workloads Through Predefined Joins

ABSTRACT

While the graph data management systems (GDBMSs) are predominant in the science world, which are usually used in a heterogeneous and complex environment, the researchers tend to use them more often in commercial and research projects. This is because graph processing is more scalable and efficient than traditional database systems. The performance of GDBMSs is affected by many factors such as the data structure and algorithms used. In this paper, we propose a graph join approach for merging predefined join queries in a relational database. This approach is implemented on a commercial database system and is used in a real-world scenario. The experimental results show that our approach improves the performance of predefined joins in relational databases by up to 70% compared to the existing methods.
Thank you & Questions?