Query Optimization in the RG Framework

CHONG FENG U4943054
SUPERVISED BY DR. QING WANG
Outline

1. Motivation
2. Background
3. RG Architecture
4. Research Problem
5. Optimisation Methodology
6. Environment
7. Evaluation
Motivation

• Graph theory has been one of the most rapidly growing research areas in mathematics over the last forty year.
• Network analysis tasks are often carried out upon relational databases.
• Data needs to be retrieved from underlying databases before applying analytical techniques on it.
• A key limitation of this model lies in the separation of managing network data and analysing network data.
Background

RG (Relation-Graph) Framework

- Provides an unified platform for **network analytics and relational data analytics**, developed by Minjian Liu under Dr. Qing’s supervision.
- Uses a **hybrid data model** (relations and graphs).
- Defines a **SQL-like query language** by extending SQL with graph constructing operations.
- Incorporates the **RG engine**, which is the core implementation of the RG Framework built upon PostgreSQL database using Python.
RG-SQL Language

The RG-SQL queries extends the traditional SQL queries with three graph operations as follows:

- Rank Operation
- Cluster Operation
- Path Operation

```
SELECT * FROM 
RANK(twitterPost, indegree) 
WHERE twitterPost is UNGRAPH as 
( 
  SELECT qid, tid FROM labelled_by LIMIT 20
);  
```
Architecture of the RG Engine

Diagram from “Towards a Unified Framework for Network Analytics”, by Minjian Liu
How to improve the performance of RG engine?

• Network analysis is often *computationally expensive*.
• The RG framework does not have any optimisation mechanisms or strategies.
• Queries are treated as *new* and *isolated* individuals each time they are submitted to proceed.
• This has introduced an issue of *inefficiency* as some of the query executions could be *repetitive and unnecessary*. 
Optimisation Overview

- **Query Optimiser:** The Query Optimiser looks up the cache to see if a query has been executed before.
- **New Query Parser:** Goes through a RG-SQL query sequentially to match and extract out both graph operations and relational sub-queries.
- **Query Tree:** The tree stores different parts of a RG-SQL query on each of its nodes.
Query Optimiser

The Query Optimiser has two main functionalities:

- **Check if a query has been executed before.**
  - Generate a hash value for each query.
  - Look up the hash value in the cache.
- **Check if any patterns in a query has been executed before.**
  - Given a constructed Query Tree, the Query Optimiser compares the tree patterns in the cache against the tree.

If cache hits, the optimizer will skip the execution process of the corresponding queries or sub-queries and reuse previous execution results.
Query Parser & Query Tree

- To facilitate tree pattern matching, a new **Query Parser** and **Query Tree** structure were developed.
- The new Query Parser goes through a RG-SQL query sequentially and iteratively to extract both relational sub-queries and graph sub-queries.
- The new Query Parser builds a Query Tree for each RG-SQL query using extracted sub-queries.
- A Query Tree stores different parts a RG-SQL query on each of its nodes.
Query Parser & Query Tree

- A RG-SQL query is made of **relational** queries and **graph** operations.
- RG query execution results can be **cached** in the memory to reduce computation for future query processing.
- A Query Tree can be compared with the cache to find reusable tree patterns.
Evaluation Environment

Datasets
- Stack Overflow Dataset:
  - 19,824,320 users
  - 7,214,697 questions

Experimental Queries
- Two queries were set up for each graph operation.
- Each query was tested using different size parameters.

Environment
- Ubuntu 14.04.4 LTS
- Python 2.7.6
- PostgreSQL 9.3.11
- Psycopg2 2.4.5

Measurement
- Time
Evaluation – Rank Operation

Result for Experimental Query 4.1

Result for Experimental Query 4.2
Thank you!

CHONG FENG U4943054@ANU.EDU.AU
0425060886