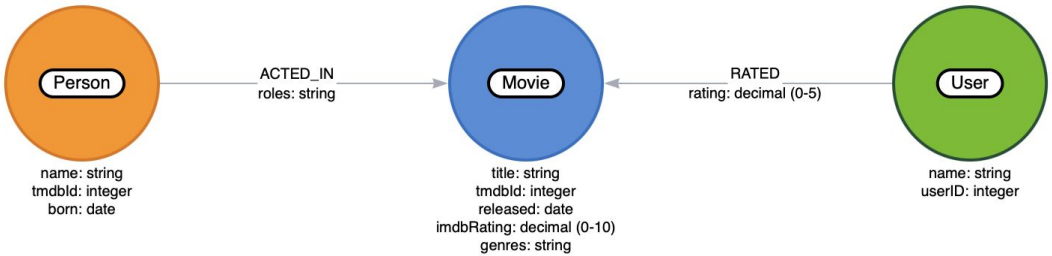
Aurendil: Accelerating Graph Pattern Matching with Physical Optimizations

Anson Yang
Neo4j Research

Introduction

- My background is in Indexing, Distributed Systems and AI4DB.
- Neo4j is a graph databases management system.
- Aurendil: We are a team in Neo4j Research.
- *Project Aim:* Designing a novel query engine based on fundamentally different technology used in current graph databases.
  - BIFROST: A Future Graph Database Runtime (2024)

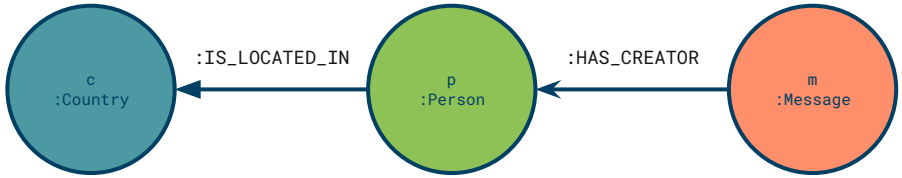


Graph Queries

- The core operation for all graph queries is Graph Pattern Matching (GPM).

  Finds all subgraphs in a larger graph that matches a given structural pattern.

```
MATCH
(c:Country)<-[:IS_LOCATED_IN]-
(p:Person)<-[:HAS_CREATOR]-
(m:Message)
```



- *Current Approaches:* Join-based or Traversal-based
  - *Join-based:*
    - Find all `(c,p)` join with `(p,m)` on `p`.
  - *Traversal-based:*
    - For each `c`, expand to `p` to form `(c,p)` and expand to `m` to form `(c,p,m)`.

The “Intermediate Result” Bottleneck

- Both approaches rely on joins (explicitly or implicitly), which result in massive intermediate tables.

- Example, query starts at country and tries to find messages in each country explodes.

- *Consequences:*
  - Memory pressure.
  - Slow execution time (from joins).

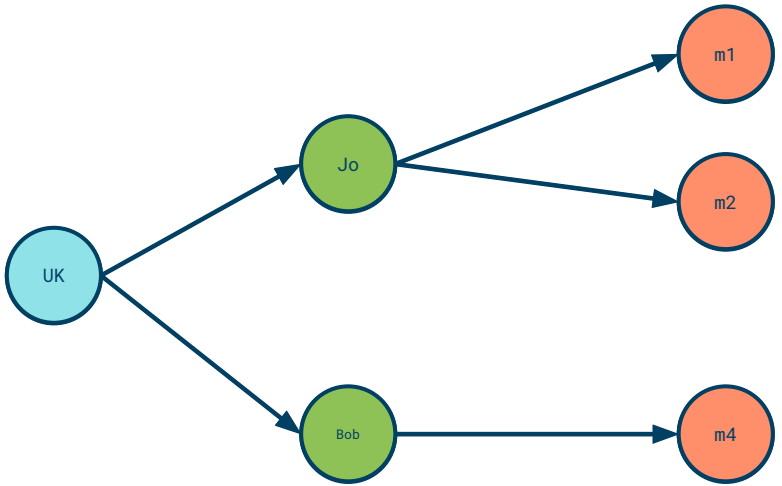
```
MATCH
(c:Country)<-[:IS_LOCATED_IN]-
(p:Person)<-[:HAS_CREATOR]-
(m:Message)
RETURN c.name, count(m)
```

For One County (UK):
Person in UK: 500,000 people
Message per Person: ~200 messages
Intermediate result size = 100,000,000

Our Approach: Search-based Execution

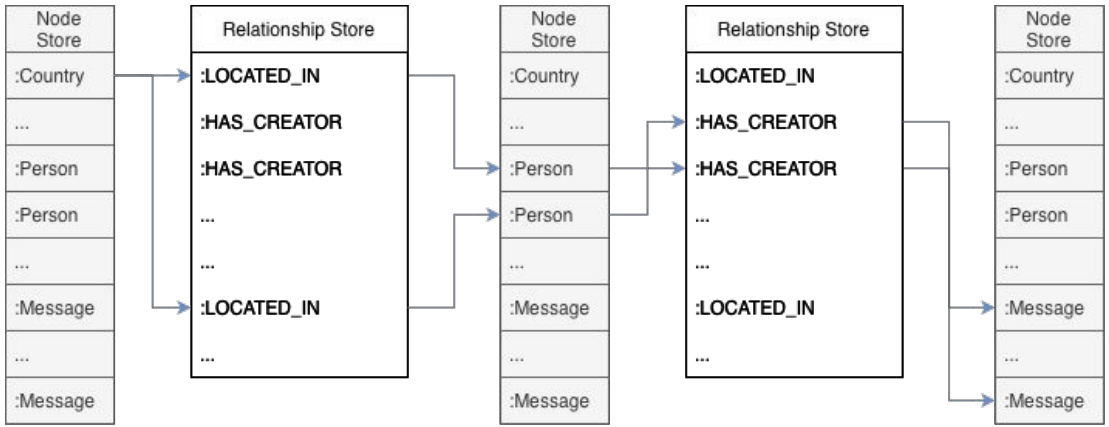
- Avoids intermediate tables and joins by streaming the paths via depth first search (DFS).

- Advantages:
  - Memory Efficiency: O(depth).
  - Early Pruning.



Physical Optimization

- Neo4j Storage Layout:
  - Node Stores
  - Relationship Stores
  - Property Stores



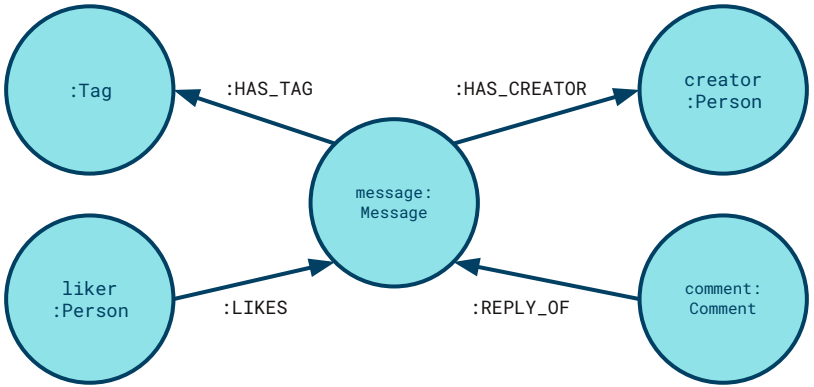
Partitioned Neighbourhood Scan: “Folds” all relationship store access into single pass.

Concrete Example #1

- *Preliminary:* We are taking queries from Linked Data Benchmark Council Social Network Benchmark (LDBC-SNB) and Labelled Subgraph Query Benchmark (LSQB).
  - Industry Gold Standards.

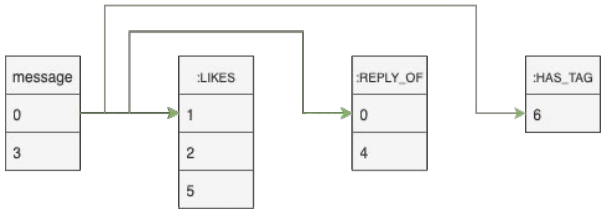
- Star-shaped query (LSQB Q4 and Q7).
  - Fold 4 relationship access around the message node into one.

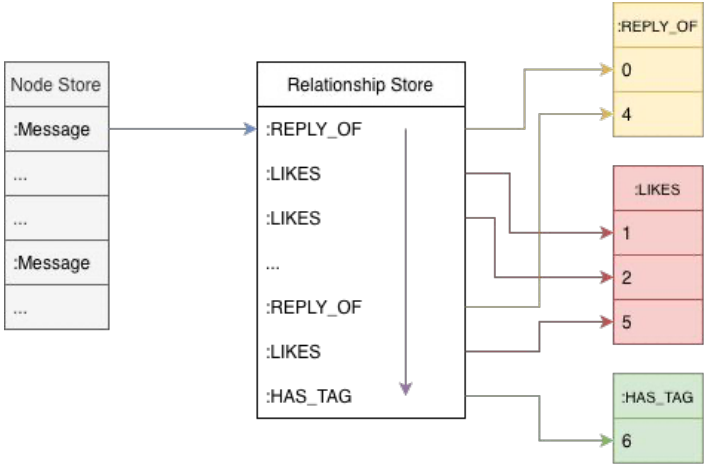
```
MATCH
      (:Tag)<-[:HAS_TAG]-(message:Message)
            -[:HAS_CREATOR]->(creator:Person),
      (message)<-[:LIKES]-(liker:Person),
      (message)<-[:REPLY_OF]-(comment:Comment)
RETURN count(*) AS count
```



LSQB Q4

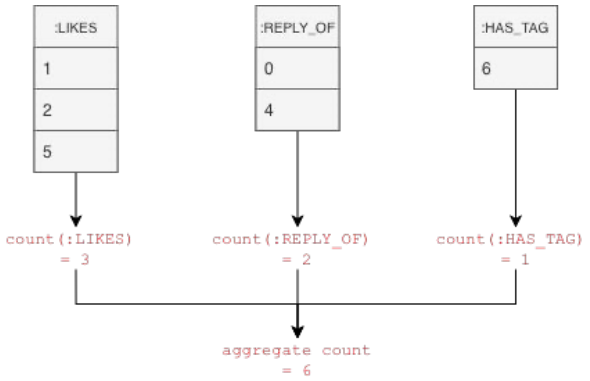
```
MATCH
      (:Tag)<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->(creator:Person),
      (message)<-[:LIKES]-(liker:Person),
      (message)<-[:REPLY_OF]-(comment:Comment)
RETURN count(*) AS count
```



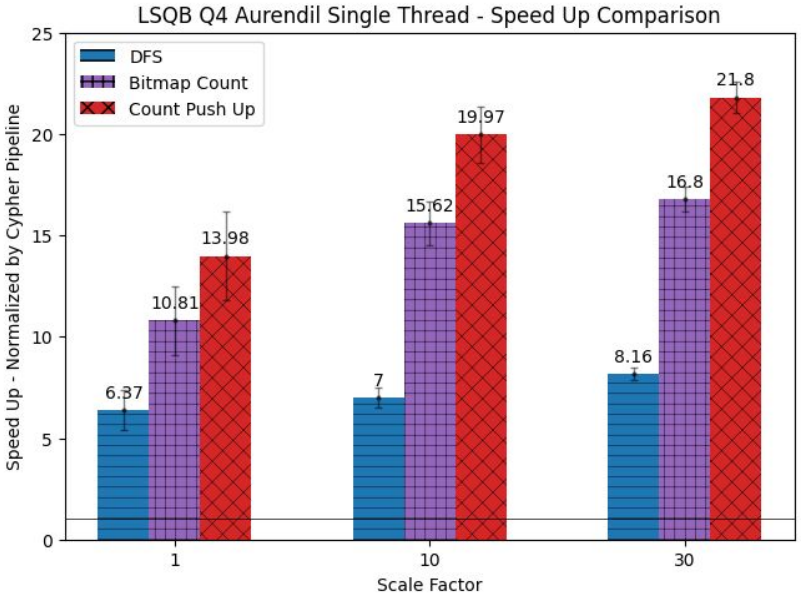


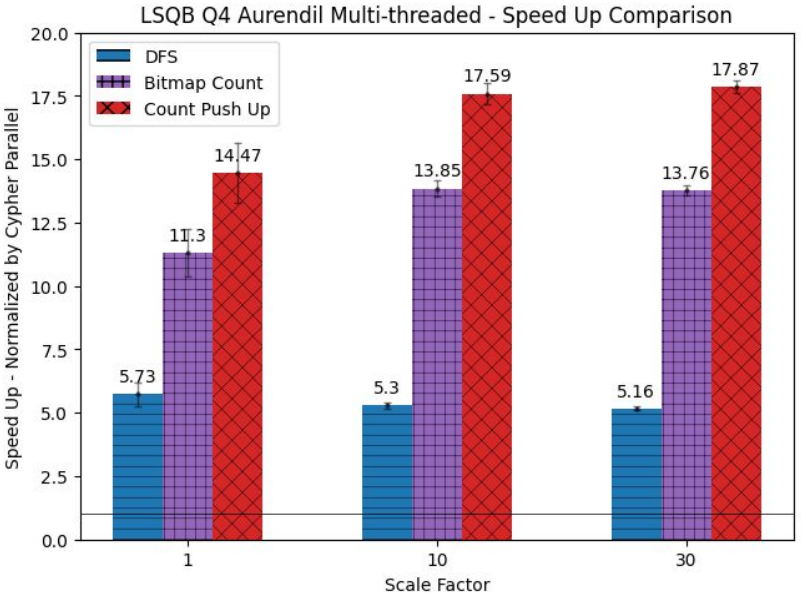
Taking advantage of the query return type

- Since the query returns a count, we don't actually have to traverse the bitmaps.
- We can aggregate up the bitmap sizes.
- In fact, when we applying folding, we can get rid of the bitmaps entirely and keep counts.



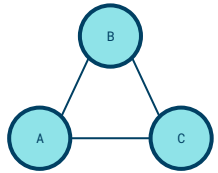
Performance Values

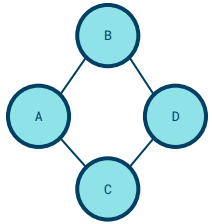




Support for Set Operations

- Another benefit of folding is that we can use the bitmaps and apply set operations (e.g., AND, OR NOT, CONTAINS).
- This allows us to reduce search depth.
- For example:
  - *Triangle patterns (LSQB Q6 & Q9):*
    - We can calculate the count mathematically using |A|x|C| (fold on B) rather than iterating through the cross-product.
  - *Diamond/Cycles (LSQB Q5 & Q8, LDBC SNB IS7):*
    - Automorphism when query pattern is symmetric.
    - Using bitmap to verify the “closing” edge without enumerating the path.





Conclusion

- *Graph pattern matching today relies on joins or traversals.*
  - This can generate large intermediate candidate sets/tables.

- *We show an alternative with DFS, search-oriented execution.*
  - One match at a time.
  - Allows for early pruning and fast backtracking.
  - Memory usage is O(depth).

- *We apply physical optimization via partitioned neighbourhood scan.*
  - Reduce random relationship store access.
  - Reduce search depth with set operations.
  - Improve execution time.

Thank you for your attention

anson.yang@neo4j.com