Summary of Research on Neo4j

Chuanyuan Chen
Department of Computer Science
Northeastern University
Silicon Valley, USA
chen.chuany@northeastern.edu

***Abstract*—In this work, I summarize my research on the selected NoSQL database, Neo4, including introduction, data model, architecture and applications.**

***Index Terms*—Neo4, NoSQL, graph, database, relationship**

I. Introduction

Neo4j is a graph-based NoSQL system. Unlike traditional Relational DBMS like MySQL, Neo4j stores nodes and edges of graphs. Fig. 1 shows a Neo4j database storing two Person nodes, a Book node, and edges labeled 'HAS-READ' and 'IS-FRIEND-WITH' indicating the relationships between nodes. Each node can contain multiple properties, just like a row can contain multiple columns in MySQL. Neo4j just fit the demand for storing datasets like social networks and knowledge graphs. Neo4j is popular and used by many organizations all cross the world including Ebay, IBM, Microsoft, NASA.

II. Motivations

Neo4j is designed to 1) easily and intuitively store and query graph datasets. 2) eliminate the need for joins, because data is stored as it's connected. 3) handle many-to-many relationships and handle deep hierarchies using graph algorithms. Fig. 2 shows the process to find interactions among 3 nodes, and Fig. 3 shows its result. 4) find patterns and hidden connections of the data. Neo4j can be easily used for recognizing patterns, generating recommendations, question-answering when combined with machine learning algorithms. These tasks are hard and not intuitive if using relational database in which all related tables must be joined together.

III. Characteristics

- Neo4j delivers **Fast Query** through native graph storage without complex joins.
- Neo4j ensures **ACID Compliance** to guarantee integrity of relationship-based queries.
- Neo4j provides **Handful Libraries** to support faster development in Java, Python, Node.js, etc.
- Neo4j designs **Scalable Architecture** to store huge amount of data.

This work was completed in part using the Neo4j AuraDB, supported by Neo4j, Inc.

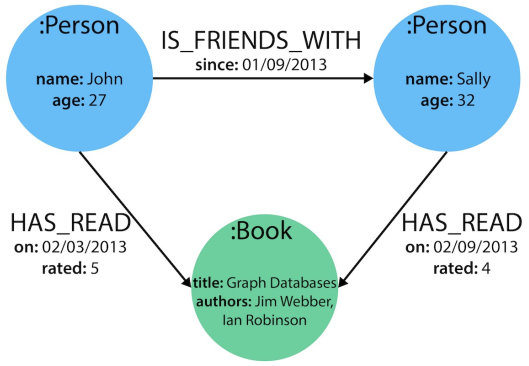


Fig. 1. A Neo4j Database.

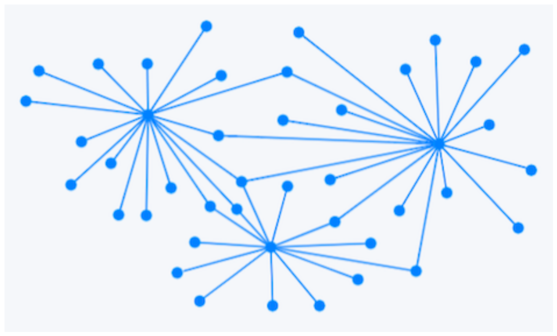


Fig. 2. Find interactions among 3 nodes.

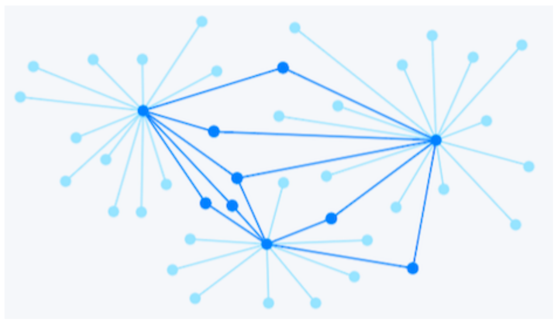


Fig. 3. Interactions found among 3 nodes.

IV. Data Model

Neo4j uses the Labeled Property Graph Model (Fig. 4): information is organized as nodes, relationships and properties. In the case of Fig. 4, two nodes are labeled 'Person', 'name' is the first property of each node. Each node points to its first relationship, a relationship contains a start node and a end node. These elements are interacted within linked list structures.

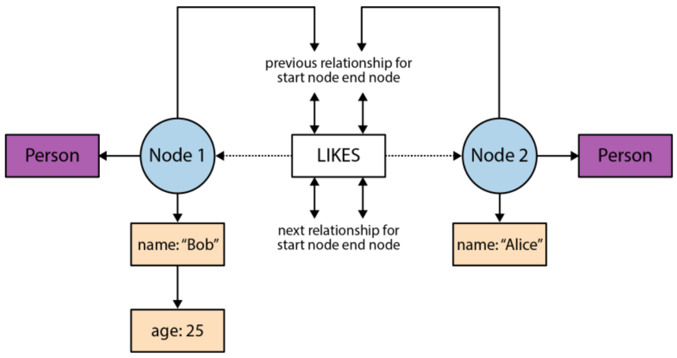
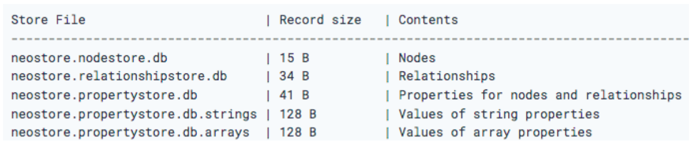


Fig. 4. Labeled Property Graph Model.

Neo4j is a native graph database: it implements a true graph model all the way down to the storage level, it is not an abstraction on top of tables. Fig. 5 shows a sample of local storage files of a Neo4j database. Nodes, relationships and properties are stored within different files.



```
Store File                          | Record size   | Contents
-------------------------------------------------------------------------------------------
neostore.nodestore.db               | 15 B          | Nodes
neostore.relationshipstore.db       | 34 B          | Relationships
neostore.propertystore.db           | 41 B          | Properties for nodes and relationships
neostore.propertystore.db.strings   | 128 B         | Values of string properties
neostore.propertystore.db.arrays    | 128 B         | Values of array properties
```

Fig. 5. Native Storage Files.

V. Architecture

Fig. 6 shows the overall architecture of Neo4j. The middle part contains similar components as other databases: Transaction Log, Caching, etc. These components are specially designed and optimized for graph database. The top level contains the graph query language, Cypher, used for CRUD operations on Neo4j databases.

VI. Applications

Here I present two python applications of Neo4j. The first illustrates basic graph CRUD operations. The second solves a shortest path problem based on Dijkstra's algorithm. The model data is shown in Fig. 7 and Fig. 8. Source code is shared at https://github.com/JulioChen10/CS7280Proj2.

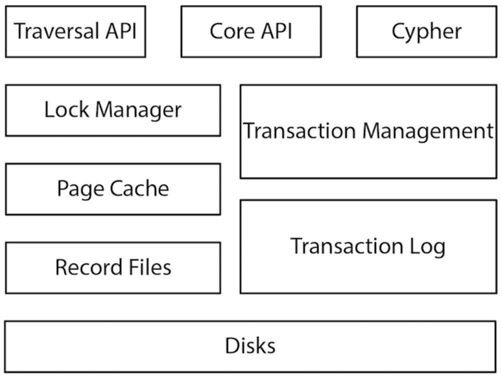


Fig. 6. Neo4j Architecture.

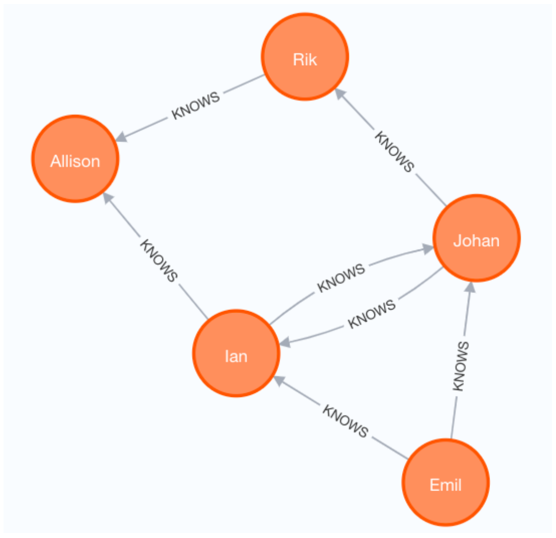


Fig. 7. Model Data of App1 (Graph CRUD).

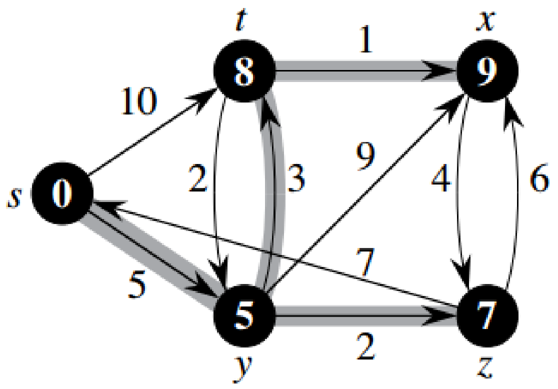


Fig. 8. Model Data of App2 (Shortest Path).

For both applications, I use GraphDatabase.driver to build a connection to the database. First, I clear the database with a DELETE statement, and then add nodes and relationships with CREATE statements. Finally, to find Emil's friends, I use this MATCH statement: MATCH (ee:Person)-[:KNOWS]-(friends) WHERE ee.name = 'Emil' RETURN friends. (Ian and Johan will be returned.) To find the shortest path from s to x, I use the Dijkstra's algorithm from the Graph Data Science library: gds.shortestPath.dijkstra.stream. ['s', 'y', 't', 'x'] is returned.

VII. Conclusion

Neo4j graph database is built on a powerful architecture, provides straightforward data modeling interfaces while keeping uncompromised performance, reliability, and integrity. Neo4j is perfect to store graph data (social network, knowledge graph) and perfect for doing analysis on them.

References

[1] Neo4j Documentation. https://neo4j.com/.

[2] Ian Robinson, Jim Webber, and Emil Eifrem. (2015). Graph Databases (2nd ed). O'Reilly.