IST 769 Research Paper

Graph Database - Neo4j

Start reading

Detailed Information:

Name: Wanyue Xiao

SUID: 720633297

Email: xwanyue@syr.edu

Date Due: March 30, 2020

Brief Introduction

NoSQL, which is the abbreviation of “Not Only SQL”, is an approach to satiate the requirements of storing large sets of distributed data, including storing a wide variety data models, capable of manipulating different types of data, and able to analyze data patterns. Due to the big data problems involving volume, velocity, and variety, the demand of performance and scalability prevails the rigidity and consistency.

As an alternative of RDBMS, NoSQL generally characterizes of having flexible schema, following BASE instead of ACID, and obeying CAP theorem (Le, 2019). Four major data storage types could be concluded for NoSQL database, which are Key Value store database (such as Redis), Column Family store database (such as Cassandra and HBase), Document store database (such as MongoDB), and Graph-based database (such as Neo4j).

Specifically, the graph database is designed to represent the relationship or properties of data. One typical instance is the Friend Connection System embedded in the social media application. Unlike other NoSQL database which utilize tables, columns, and documents for data structuring, graph-based dataset uses nodes to reflect entity and uses edges (or lines) to indicate relationships. Any comments related with nodes, or the entity, could be viewed as the properties of that entity.

As an open-sourced, NoSQL, native graph database which was issued in 2010, Neo4j now supports functions of graph analytics, data integration, and graph visualization (Neo4j, n.d.).

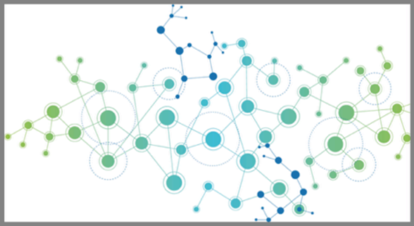
Key Features

Despite of supporting property graph models like other graph database does, Neo4j has several features that differentiate it. Some of those features will be discuss separately in later sections:

- **Whiteboard-friendly Data Modeling**: Even the non-technical personnels are capable to participate in data modeling process.
- **Constant Time Transaction**: Unlike other databases, Neo4j follows the ACID transaction which guarantees the data reliability, consistency, and integrity while still maintains tremendous flexibility and agility.
- **Next Generation Query Language**: A declarative SQL-Like language called Cypher has implemented in Neo4j which aims to optimizes graphs
- **Developer Friendly**: Relied on its drivers, Neo4j can be integrated with various prevailing programming languages, including Java, JavaScript, Python, PHP, .Net, Go, and Ruby.
- **Online Browser Interface**: Despite a desktop application which support local instance management, Neo4j also has online browser interface which enables user to query and visualize data in the database via Cypher query language.
- **Clustering Options**: Neo4j provides two types of clustering which are the high-availability clustering and causal database.
- **Cache Sharding**: Not like other NoSQL database using partitioning techniques, neo4j is not partition tolerant. It supports cache sharding yet not database sharding.

Storage Requirment

Neo4j could be installed in different environments, including the popular MacOS, Windows, and Linux operation system. The prerequisites vary significantly depends on the usage of software (Neo4j, n.d.).
For personal user: the CPU should be at least Intel Core i3 while Core i7 is being recommended. Besides, the system memory should have at least 2GB as a minimum while 16GB or more is being recommended. As for storage, 10GB SATA is the minimal requirement while SSD with DATA express is being recommended.
For server-based environment: the CPU should be Intel Xeon processors. The system memory should have at least 8GB as the minimum. For storage, it demands SATA i7.2K RPM 6Gbps Hard Drive as the minimum.
For cloud environment: the CPU should be at least 2vCPU. The system memory should have at least 2GB as the minimum. For storage, it requires at least 10GB block storage as the minimum.



Applying ACID Principle

The CAP theorem suggests that database development generally encounters three data issues: consistency, availability and partition tolerance. Even though Neo4j belongs to NoSQL, it is not a distributed database which obeys BASE principle, it follows ACID principle (Atomicity, Consistency, Isolation, Durability) like RDBMS does.

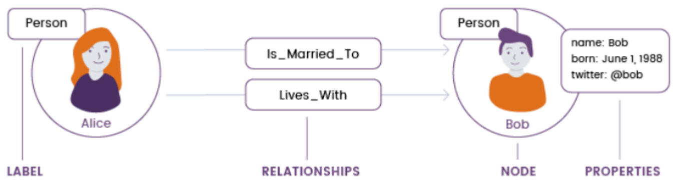
Hence, Neo4j has a problem in scaling out even though it provides method to partially scale out using techniques like cache sharding. In this case, the developer can use both the Neo4j and Hadoop and treats each of them as a compliment. Besides, the casual clustering enables Neo4j to tolerant faults (Neo4j, n.d.).

Technical Limitation

1. **Sharing Issue**: developer could not share Neo4j database since "the whole dataset in ONE server and only vertical scalability is possible" since Neo4j does not support sharding or partition (Miller, 2013).
2. **Size Limitation**: Due to the upper bound limit for the graph size, it is difficult for Neo4j to hold tens of billions of nodes, properties, and relationships in a single graph (Miller, 2013).
3. **Security Issue**: No security is provided at the data level, including data encryption and security auditing (Miller, 2013).
4. **Language Conversion Issue**: Due to the schema-less nature, Neo4j might show different results for running the same query on different clients which use different language (Armbruster, 2016).

Property Graph Model

As mentioned above in the introduction, the three key components of a property graph model are nodes, relationships, and properties (Neo4j, n.d.).



- **Nodes**: nodes are primarily the entities which could be represented by key-value pair (attribute). The "Alice" and "Bob" are the entities. Here the node could be tagged with zero or more than one labels which indicate either the roles in the domain or metadata attached to that node. In the domain, node with same label belongs to the same set. For example, all nodes indicating employee could be labeled with the **label :Male**, **:Female** and **:Person**.
- **Relationships**: relationship is a directional, semantically relevant link between two nodes. In the example, the relationship between entity company and city is "**:Lives_With**". The relationship line also has at least one attributes if required.
- **Property**: property, or the attribute, is the characteristics of a node or a relationship. The "**name:Bob**" and "**Born:June 1, 1988**, " below the "Bob" node is the attribute of that node.

RDBMS Table convert to Property Graph Model

Property graph model is easy to comprehend even for developer who has basic knowledge since one can transform the tabular data model to graph based model. The difference is that RDBMS rely index lookups and joining functions to connect data that stored in different entities. As a schema optional model, property graph model just grabs all those data (or normally each row) from those entities and links them together. Below is a cheat note from Neo4j official document (Neo4j, n.d.):

- *Table to Node Label* – each entity table in the relational model becomes a label on nodes in the graph model.
- *Row to Node* – each row in a relational entity table becomes a node in the graph.
  Column to Node Property – columns (fields) on the relational tables become node properties in the graph.
- *Business primary keys only* – remove technical primary keys, keep business primary keys.
- *Add Constraints/Indexes* – add unique constraints for business primary keys, add indexes for frequent lookup attributes.
- *Foreign keys to Relationships* – replace foreign keys to the other table with relationships, remove them afterwards.
- *No defaults* – remove data with default values, no need to store those.
- *Clean up data* – duplicate data in denormalized tables might have to be pulled out into separate nodes to get a cleaner model.
- *Index Columns to Array* – indexed column names (like email1, email2, email3) might indicate an array property.
- *Join tables to Relationships* – join tables are transformed into relationships, columns on those tables become relationship properties

Query Language: Cypher

Neo4j primarily supports two types of query language. The first is **Gremlin** which is a Domain Specific Language implemented on top of Groovy programming language (Neo4j, n.d.). The another called **Cypher** is a SQL-Like language which could be used to express the fundamental graph elements (that are nodes, relationships, and property) (Neo4j, n.d.). Given that graph model mimics the intuitive way human understand information, cypher is designed to be self-descriptive and to help developer for developing simple yet logical graph.

The syntax of cypher is pretty straightforward:

- **CREATE ()** # One can use the CREATE function to get an anonymous node which can be refer to any node in the database.
- **CREATE (p), (e)** # One can create multiple nodes at one time. Here the 'p' and 'e' are variables.
- **CREATE (p:Person)** # Inside the parenthesis, label(s) could be stated to represent the node type. Specifically, labels have colons (:) in front of them. One can also create node that has label but does not have variable.
- **CREATE (p:Person {name: 'Bob', born: 'June 1, 1988', twitter: '@Bob'})** # In addition to labels, a node can have properties which is in JSON format.

After creating the entities, **we can create relationship to depict the connection between entities**.

- **CREATE (p:Person {name: 'Alice'})-[:Lives_With {since: 2018}]->(p:Person {name: 'Bob'})** # Using the arrow '-->', '<--', or even the undirected line '--' to represent the connection. Inside the arrows, one can also add relationship property inside.

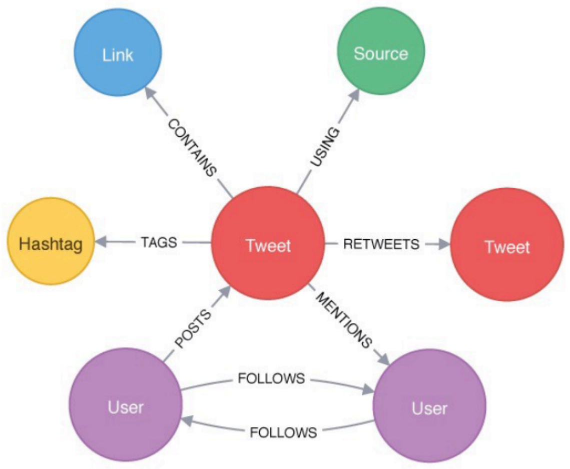
Cypher also has other keywords:

- **MATCH (p:Person)** # Match searches for an existing node, relationship, label, property or pattern inside the database, which similar with SELECT statement in SQL.
- **RETURN (p)** # Using return statement to retrieve data.

Above is the example of finding Person nodes in the graph and return those nodes via RETURN statement. In more complicated scenario, the developer can also add more constrain to the querying, such as indicate the node property or relationship property.

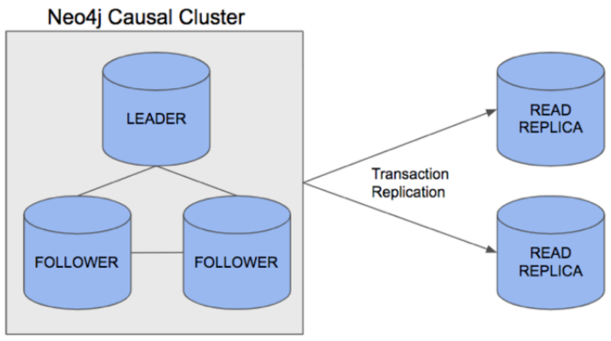
Here is another exmaple:
MATCH (p:Person {name: 'Alice'})-[:Lives_With]->(p:Person {name: 'Bob'})
RETURN p
In this case, the system will find Alice's Person Node and another Person node with property "name: 'Bob'" that she connected to. To do that, the system will follow the ':Lives_With' relationship from Alice's Person Node to Bob's Person node. Then one can use return statement to extract variable p from the database. For more detailed information, please to go the official website Neo4j Cypher Query Language.

Below is a visualization of Cypher Query Language:



Available Clustering Option

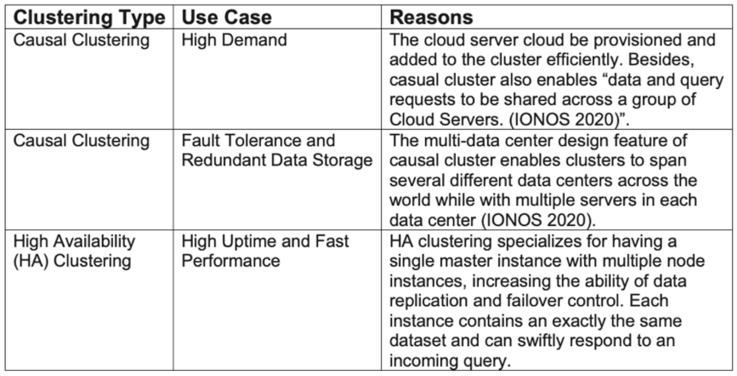
Even though those two clustering both maintain the operation excellency in performance since Neo4j can distribute workloads across different servers, they still have some distinct advantages respectively (Neo4j, n.d.). For example, for client who wishes to store a large volume of dataset, causal luster is the most appropriate solution due to its scalability. However, HA cluster option had been totally removed from the most current edition. Hence, this report will only introduce the causal clustering.



Cluster architecture

As this graph illustrated, each casual cluster consists of at least three instances which take different cluster role. Every instance can be either a leader that is responsible for cluster communication and data writing or a follower that helps scale the read workload ability (Allen, 2019). By using a consensus protocol (RAFT), each database has a complete copy of the entire data.

Below are some typical use cases:



Clustering Type	Use Case	Reasons
Causal Clustering	High Demand	The cloud server cloud be provisioned and added to the cluster efficiently. Besides, casual cluster also enables "data and query requests to be shared across a group of Cloud Servers. (IONOS 2020)".
Causal Clustering	Fault Tolerance and Redundant Data Storage	The multi-data center design feature of causal cluster enables clusters to span several different data centers across the world while with multiple servers in each data center (IONOS 2020).
High Availability (HA) Clustering	High Uptime and Fast Performance	HA clustering specializes for having a single master instance with multiple node instances, increasing the ability of data replication and failover control. Each instance contains an exactly the same dataset and can swiftly respond to an incoming query.

Appropriate Use Cases

- **Real-time Recommendation System**

A good recommendation engine is crucial for online business websites, especially e-commerce. Since graph database has the advantages of express connected data, Neo4j is suitable for handling real-time recommendation information. According to the paper of Miller (2013), there are two correlation methods applied the graph database when it comes to recommendation engine. One is the **'item-to-item'** correlation recommends products based on associations between new items and items that user viewed or purchased previously. This is also called 'content-based filtering'. Another is **'user-to-user'** recommendation system, which is also known as collaborative filtering, that able to generate predictions based the correlation or connection made by looking at how the user interacts with the system.
In this case, only graph database has the merits of retrieve relationships efficiently according to user real-time behaviors, such as purchase records, interaction, and product reviews. By using Neo4j, the database can store, track, and query the relationships stored inside the database and further analyze the pattern based on the extracted recommendation data.
Example: Amazon's Item Suggestion Engine and Ebay Recommendation System

- **Knowledge Graph**

Except storeing domain knowledge, knowledge graph could also being used to provide contextual search solution to search engine. Neo4j enhances the search capabilities to deliver relevant results through improving simple keyword search and providing additional results related to keywords (Gleb et al., 2018).

Example: Facebook's Friend Management System

- **Social Networks**

Similar with recommendation system, social networkd records connected between people. Neo4j is perfectly suitable for storing social network data, which not only shortens the period of application development but also enhances the application's overall performance. (Gleb et al., 2018; Cattutoet et al., 2013)

Example: Facebook's Friend Management System

- **Fraud Detection**
- **Master Data Management**
- **Network and IT Operations**
- **Identity and Access Management**

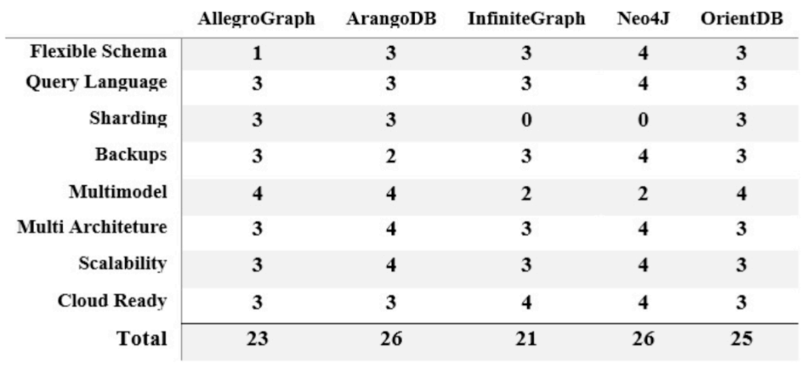
Inappropriate Use Cases

Considering graph database are naturally indexed by relationship, it is a good idea to use graph database to record a lot of relationships, which is both flexible and declarative. Hence, neo4j is suitable when inter-connections between data need to change frequently. Still, it has limitation when it comes to manage a gigantic list of data. One typical example is customer transaction data storage. RDBMS is a better option in this scenario.
Aside of that, graph database is not that useful if used individually for optional use cases since this type of database are not designed to process high volumes of transactions efficiently or to query data that span the entire database (Rend, 2017). The best bet is to combine the graph database with other RDBMS or NoSQL databases.

Competing Products & Underlying Reasons

Except Neo4j, there are many powerful graph databases available, such as AllegroGraph, ArangoDB, InfiniteGraph, and OrientDB. In this section, the characteristics, advantages, and disadvantages of those graph databases will be summarized and compared.

According to the research of Fernandes and Bernardino (2018), they made a comparison table valued from seven dimensionalities in terms of flexible schema, query language, sharding, backup, scalability, and etc. In each dimension, the score scale from 0 to 4. **4 points represent 'Great', 3 stands for 'Good', 2 indicates 'Normal or Average', 1 means 'Bad' while 0 means 'does not support'.**



	AllegroGraph	ArangoDB	InfiniteGraph	Neo4J	OrientDB
Flexible Schema	1	3	3	4	3
Query Language	3	3	3	4	3
Sharding	3	3	0	0	3
Backups	3	2	3	4	3
Multimodel	4	4	2	2	4
Multi Architeture	3	4	3	4	3
Scalability	3	4	3	4	3
Cloud Ready	3	3	4	4	3
Total	23	26	21	26	25

From the table, one can see that the graph database which have the highest total score are ArangoDB and Neo4J. Then it should be OrientDB.

- **Neo4j** has an excellent performance on flexible schema, query language, scalability, cloud ready. However, the main weakness is that it does not support sharding or partitioning (except cache sharing which is different).
- **ArangoDB** is a schema-free database which allows flexible data storage. It has its unique and powerful SQL-Like query language called AQL which "enables really impressive complicated queries and processes to be pushed to the backend" (Fernandes & Bernardino 2018).
- **ArangoDB** supports sharding and could be implemented in AzureCloud.
  OrientDB is a distributed graph database which is capable of working in a schema-less, schema-full, and schema-hybrid solutions. It also supports both SQL and gremlin. Unlike Neo4j, OrientDB supports sharding and could be implemented in cloud environment.

Then if developer wish to find a graph database which has the ability of data sharding, it can consider ArangoDB or OrientDB.

If the developer cares more about the query efficiency and scalability, Neo4j could always be a good choice.

Reference Lists:

- Allen, D. (2019, March 1), *Querying Neo4j Clusters*, https://medium.com/neo4j/querying-neo4j-clusters-7d6fde75b5b4.
- Armbruster, S., (2016, Feburary 24), *Welcome to the Dark Side: Neo4j Worst Practices (& How to Avoid Them)*, https://neo4j.com/blog/dark-side-neo4j-worst-practices/.
- Cattuto, C., Quaggiotto, M., Panisson, A. & Averbuch, A., (2013), Time-varying social networks in a graph database: a Neo4j use case. *In First international workshop on graph data management experiences and systems* (pp. 1-6).
- Fernandes, D. & Bernardino, J., (2018), Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. *In DATA* (pp. 373-380).
- Gleb, B. & Vlad, L., (2018, July 05), *Capabilities of the Neo4j Graph Database with Real-life Examples*, https://rubygarage.org/blog/neo4j-database-guide-with-use-cases.
- IONOS. (n.d.), *Use Case: Neo4j Causal vs High Availability Cluster*, https://www.ionos.com/community/markdown-migration/use-case-neo4j-causal-vs-high-availability-cluster/.
- Miller, J. J., (2013), Graph database applications and concepts with Neo4j. *In Proceedings of the Southern Association for Information Systems Conference*, Atlanta, GA, USA (Vol. 2324, No. 36).
- Neo4j. (n.d.), *A Graph Platform Reveals and Persists Connections*, https://neo4j.com/product/?ref=home.
- Neo4j. (n.d.), *Model: Relational to Graph*, https://neo4j.com/developer/relational-to-graph-modeling/.
- Rend, B. (2017, March 14), *The Good, The Bad, and the Hype about Graph Databases for MDM*, https://tdwi.org/articles/2017/03/14/good-bad-and-hype-about-graph-databases-for-mdm.aspx.