A Survey of Neo4j database-Doing Movie Statistics Analysis

Junteng Ma
Concordia University
Montreal, Canada
thehappyboy_m@outlook.com

Zhangwen Yan
Concorida University
Montreal, Canada
zhangwen.yan@mail.concordia.ca

ABSTRACT

Graph database, being a kind of NoSQL database, is able to present relationships between entities. Neo4j is a Java-based graph database which can hold big data and provide cluster services. In this project, we will use the Neo4j to hold and analysis big data, and deploy cluster to study its distributed aspects. In this report, we will discuss the charactetistics of Neo4j graph database, discussing clustering and distributed features and give our data analysis result using Neo4j.

keywords: Neo4j, graph database, cluster, distributed system

1 INTRODUCTION

In many real-time applications, relationships play an important role. Tradition SQL database handle relationships between entities by inner joinning tables which could cause wastage of memories. As a result, relation databases appears to handle data based on the relation model of data. In the field of NoSQL, graph databases have become the database type with the most attention and trends in the last decade. A graph database is a method of connecting data using graph theory. Connecting pieces of data, called nodes, with relationships between nodes, and allows a structure that can be used to store, query, and represent data. Storing data, presenting it, and deriving conclusions in graph form is not only very interesting but also high performance.[7]

Neo4j is a high-performance graphical database that describes the complete situation of any user through nodes, relationships, and properties. The nodes can be added or deleted at any time, thus effectively solving the problem of memory wastage when storing semi-structured and unstructured data, and by its unique deep traversal interface, it can quickly find nodes of any depth under different relationships.[5]

In this project, we try to make a survey on the ability to deal with relationships among data. Firstly, we try to understand the architecture, data presentation and Cypher SQL to fetch data in Neo4j. We also make a survey on clustering and distributed aspects in the database. We have made experiments using a real movie dataset[3] and we will present some results of out project.

2 NEO4J SYSTEM

2.1 Data presentation and storage

The basic data model in Neo4j consists of nodes and relationships. Graphs databases store data in nodes and relationships in the form of property.[4] A node is usually used to represent an entity, such as an objective object in the real world, or an abstract event. node is the smallest unit that makes up a graph.At the same time, nodes and Relationships contain property in the form of key/value. Nodes are connected by relationships defined by relationships, forming a relational network structure. Labels are used to classify nodes. In Neo4j, it is possible to query a particular node based on different label types, and a node can have multiple labels. Fig.1 shows what a neo4j database looks like.

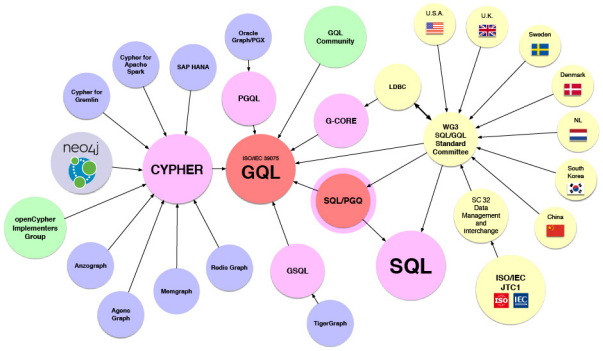

Figure 1: An example of data in Neo4j

Each node record contains the ID of its first property and the ID of the first edge in the relationship chain, and when reading the property of a node, we can traverse along a one-way chain structure starting with the pointer to the first property. When querying the edges associated with a node, we can start from the pointer to the first edge in the node, and after getting the first edge we can traverse along the bi-directional edge table associated with the specified node (start or end node) until we find the edge we are interested in. After finding the specified edge record, we can use the same one-way linked list structure as the node property to read the edge property, or it can check its starting and ending node records associated by ID, which can be multiplied by the length of the node record to quickly derive the offset of each node in the storage file.

It is worth noting that the relationships in the graph database are directional in nature. In the graph, we not only know which items are related to which property, but we know how they are related, and there are no restrictions on what can be related and how.

Relationships are stored using a point-cut approach, where information is complete for each edge and a Node is cut with its information redundantly in different edges.[4] By default, Neo4j's storage files are located in the NEO_HOME\data\databases\graph.db directory, and the storage files will be generated the first time Neo4j is run. Neo4j is a schemaless database, it use a fixed record length to persist data and track offsets in these files to understand how to fetch data to answer queries.

Neo4j's individual storage files remain largely consistent in form, the node, relation, label and property of the property graph

are stored separately in different files. The node data is stored by the main storage file, the label-related storage file and the attribute-related storage file. The main node storage file is neostore.nodestore.db, a fixed-length type of record file, each Record representing a node. The storage of attributes is done by the main storage file, the attribute key-related storage file, and two dynamic record format files that store longer characters and arrays in the attribute values. The long string value storage file name is neostore.propertystore.db.strings, and the array value storage file name is neostore.propertystore.db.arrays. The main property store file name is neostore.propertystore.db, and the length of the property Record is the longest in the fixed-length record format, at 41 bytes.

2.2 Cypher

Creating a Neo4j database is as simple as adding nodes, relationships, and their properties to the database. Neo4j has Cypher as a query language, a simple and intuitive graph language, which is constructed based on English words and simple diagrams. Cypher uses the permutation of text to reduce the visualization of graphs, making queries easy to write and read. Below is a comparison of the graph and the cypher language.[6]

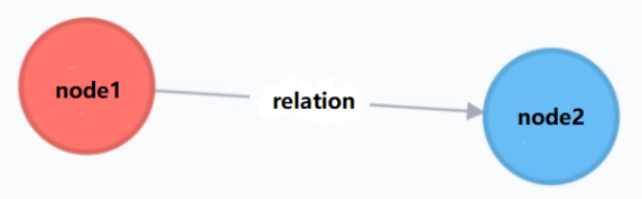

Figure 2: Data presentation in Neo4j

Fig.2 can be expressed in cypher language as (node1{property})-[:relation]->(node2). () means the node, [] means the relation, {} means the property, > means the direction of the relation. Nodes and relationships are connected with -. This is in line with both the intuitive thinking of human language and the presentation of graphs. Cypher can be used to search for specific data, construct specific subgraphs and apply operations such as counting or averaging. It also allows manipulation of node properties through numeric and string operations. queries in Cypher are oriented towards two commands; MATCH and RETURN. The MATCH is either a query for the desired node in the graph or a relation, RETURN gives the result of the final query. Within this scope, Cypher can also perform boolean, string, and numeric operations in a manner similar to SQL. Cypher also contains clauses for writing, updating, and deleting data. CREATE and DELETE are used to create and delete nodes and relationships.

2.3 Driver and bolt protocol

The Neo4j Browser is a developer-oriented tool for the Neo4j database. The Neo4j Browser can be used to add data, run queries, create relationships, etc. using Cypher statements (described in 2.2). It also provides an easy way to visualize the data in the database. We used the Neo4j Browser as a platform for presenting results in our subsequent experiments.

The Bolt Network Protocol is a client-server protocol for database applications that is both speedy and lightweight. Clients can transmit statements, which are made up of a single string and a collection of type arguments. The server responds to each request using a result message and, optionally, a result log stream. TCP or WebSocket connections with optional TLS encapsulation are used in all versions of the Bolt protocol. The default port is 7687, which is the recommended value.[8]

Neo4j provides a powerful client-side driver that can automatically handle LEADER node changes for neo4j clusters (described in part 3 Cluster below), load-based request routing, and the flexibility to customize routing policies based on business requirements.

3 CLUSTER

3.1 Overview

Neo4j provides clusters which includes a set of instances each naming a cluster. Each cluster have a unique name and are deployed under unique addresses. Neo4j defines different roles in the cluster, such as primary servers and secondary servers. The primary servers can include one single instance or multiple core instances and can provide Read-Write authority. Secondary servers, including read-replicas can only provide read authority. In a real project, leader and follower role are assigned to primary servers while read-replica are assigned to secondary servers. With these roles, the casual clustering of Neo4j can provides features such as safety, scale and causal consistency. Details about primary servers and secondary servers will be described in following sessions. [2]

3.2 Primary servers

Primary servers is servers with read-write authorities. It is the key part to maintain casual consistency in the system. There are two types of primary servers based on number of machines in the primary servers.

The first type in single instance which means the cluster containing only one single server which allow write and read operations. However, in this type, several secondary servers can be included. This type can provide casual consistency as only one instance is permitted to write into the database. Redundancy is only provided in the secondary servers. There is only one single instance in primary server, meaning that if this server break down, no write operations will be provided. As result, this type cannot provide fault tolerance.

The second type is clusters including multiple core instances. To maintain consistency, every transaction will be duplicated to all instances by Raft protocol. This type of primary server can provide fault tolerance as there are multiple servers which have read-write authority. Meanwhile, casual consistency is also maintained by the Raft protocol.

3.3 Raft protocol

It is important to ensure the consistency among servers, meaning that if a transaction is committed to one server, the transaction should also be committed to other servers with the same role. Raft protocol is a protocol maintaining the consistency. In this protocol, if one transaction is accepted by over half ($> \frac{N}{2}$) of instances in the cluster, this transaction will be permitted and executed.

Considering a cluster of $2f + 1$ instances, even if f instances break down, there will also be $f + 1$ servers accept the transaction, which is bigger than $\frac{N}{2}$. This means that the Raft protocol ensure the writing operations on $2f + 1$ servers will not be effected if f servers break down.

3.4 Secondary servers

Secondary servers is a set of instances only providing read operations. It is also called Read Replica instances. The transaction in Core Servers will be replicated to Secondary servers. They will periodically send requests to primary servers and fed data. Normally, Read replicas should be run in a relative larger numbers. The responsibility of secondary servers is to seperate scalled out read workloads.

Particularly, secondary servers can be a replicated server of primary server if it is a single instance. The Read Replica can be set to a single instance if the single instance is no longer available.

3.5 Life cycle

This section will introduce the lifecycle when a cluster is deployed. When setting a instance, a set of discovery addresses will be assigned to this instance. It will firstly contact these servers and ask to join. For read-replicas, they do not need to care about the Raft protocol. They only need try to handshake with other listed servers. However, if a core membership join the group, to maintain the Raft protocol, every membership need to update their Raft group. [1]

In the Raft protocol, there are three roles that an instance can be playing: Leader, Follower, and Candidate. For LEADER election, there are the following steps: first Neo4j supports an automatic recommendation of LEADER, a core server is recommended by the cluster as a LEADER. When the current LEADER fails, the remaining nodes will automatically elect a slave as a LEADER. In addition to the LEADER node, the remaining core will become FOLLOWER. This high availability model can only be achieved by a load balancer. Only the core server with the latest logs is eligible to be a LEADER, and the LEADER will always be given priority to perform log replication operations. This sequence of operations will ensure the consistency and security of the logs.

The protocol for read-replica periodically update their data is called catch up protocol. To be noticed, Neo4j transaction IDs are strictly monotonic integer values.If Read Replica transaction history is largely different from core servers, the shipping will not be done. The read-replica will directly copy from the core servers.

4 DISTRIBUTED FEATURES

4.1 Consistency

Causal consistency, i.e., the user should be able to read the data it has previously written to the database. For a standalone mode database, causal consistency is always guaranteed, just make sure the transaction that wrote the data has been committed and the read and write data can be checked against each other. However, in clustered mode, there may be data latency due to read and write operations spanning neo4j processes and the read-only replication and core nodes being asynchronous. neo4j is able to achieve causal consistency for any two transactions, even if the transactions are executed across nodes. This is because when using causal clustering, neo4j chains transactions to ensure causal consistency. When executing a transaction, the client can request a bookmark, which is marked with a location in the transaction as a parameter. The bookmark is automatically passed between transactions to ensure correct write-first, read-later semantics from the client's perspective. This means that for any two transactions, it is guaranteed that the second transaction will only start after the first transaction has been successfully committed, neo4j guarantees that among multiple transactions in a session, different transactions will be executed in causal order.

4.2 Fault tolerance

This section will introduce how Neo4j perform fault tolerance in terms of core shutdown and read-replica shutdown.

When a clean shutdown occurs in a read-replica, it will send discovery protocol to clusters to ask them remove it in the overview of the cluster. For an unclean shutdown, the core server will discover the instance offline when it send request to the instance. The discovery machine will initially hide the replica in the entry, until the read-replica in back online.

When a clean shutdown occurs in a core server, it will be handled by raft protocol. If a core server shutdown, every other members in the clusters will not be able to contact this server. It will be eventually voted out from the Raft group.

However, the cluster will fail to proceed if $\lfloor \frac{N}{2} \rfloor$ servers shutdown in a short period of time. The vote will not be able to be held if over half of servers shutdown. If members are gradually lost, the Raft protocol have time to reduce the membership.

5 RESULTS

5.1 Load data

In this system, load data is proceeded by neo-admin. Neo-admin is able to initially insert a large amount of data into the database in a short period of time. In this project, we have 1331825 nodes and 2896008 relationships, and these are inserted into database within 1 min.

The command of inserting data is shown as follows:

```
neo4j-admin import --nodes movies.csv
genres.csv users.csv --relationships tags.
    csv ratings.csv movieGenres.csv
```

and we have example of inserted Node and relationships as:

Node:

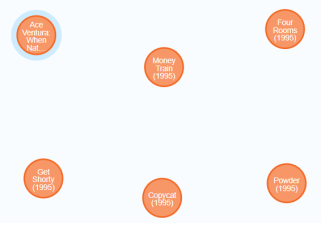

Figure 3: Example nodes of movies

This is an example of nodes belonging to movies. Movies have properties such as id,title.

Figure 4: Example nodes of genres

This is an example of nodes belonging to genres. Genres have properties of name. Relationship:

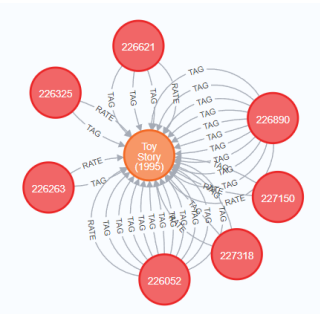

Figure 5: Example relationships of tags

A tag relationship starts from a user to a movie, meaning a user have tagged this movie with a tag.

5.2 Statistics

Fig.9 shows the interconnected Movie nodes and Genre nodes (limited to 25 relationship edges) based on the GENRE relationship. The blue node is Genre, and each different Genre node represents a different movie category. The orange node is Movie, and each Movie node represents a separate movie. We can see that a movie always has one or more genres, such as the movie "Heat (1995)" connected with "Crime", "Action", "Thriller" three Genres. The movies that fit the "Action" genre are "Heat (1995)", "Sudden Death (1995)" and "GoldenEye (1995)". According to this figure, it can clearly understand what genre each movie belongs to and what kind of movies are available under each genre category.

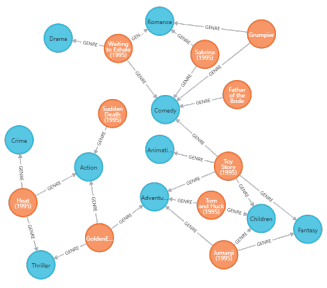

Figure 6: Example nodes of genres

Graph databases excel at processing enormous volumes of complicated, interconnected data, and we may readily query the relationship between two nodes when we want to know about another node that has a link with one we already know about. For example, finding a list of movies that both userId 1 and userId 250 have rated is a very useful query. Finding the movies users want to watch in a large volume of data is often an immediate need for users.

```
MATCH (u1:User)-[:RATE]->(n:Movie)<-[:RATE
    ]-(u2:User)
WHERE u1.user_id CONTAINS "1" and u2.user_id
     CONTAINS "250"
RETURN DISTINCT n.title LIMIT 15
```

	n.title
1	"Dark Knight, The (2008)"
2	"In Bruges (2008)"
3	"Bourne Ultimatum, The (2007)"
4	"Simpsons Movie, The (2007)"
5	"Death at a Funeral (2007)"
6	"Sicko (2007)"

Figure 7: Movies rated by two user

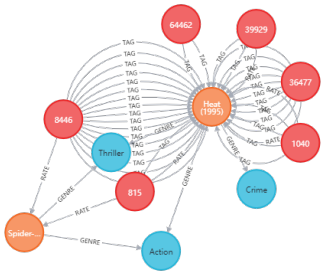

Figure 8: Movie with the same tag and genre as "Spider-Man"

```
//similar movie with spider-man base on
    GENRE and TAGx3
MATCH p=(m:Movie {title: "Spider-Man (2002)
    "}) -[:TAG|GENRE|RATE*3]-() RETURN p
    LIMIT 15
```

Due to the increasing number of movie profiles, it is difficult for users to choose what movies they want to watch. An effective solution is to suggest movies that users may be interested in. Therefore, we use a graph database of connections to recommend movies. Here

is an example, based on the movie "Spider-Man (2002)" that the user has seen, the movie with the same tag and genre is "Heat (1995)". Both movies link to the same relation of Genre and multiple users give the same tag, so we can say that the two movies are similar.

If the user selects movies by genre alone, the Cypher query below will return the movies with the most similar tags. Figure is the movie with the most similar tags to "Spider-Man (2002)".

	rec.title	genres	commonGenres
1	"40 Days and Nights (2012)"	["Adventure", "Action", "Sci-Fi", "Thriller"]	4
2	"InAlienable (2008)"	["Adventure", "Action", "Sci-Fi", "Thriller"]	4
3	"Jurassic World: Fallen Kingdom (2018)"	["Adventure", "Action", "Sci-Fi", "Thriller"]	4
4	"Airwolf: The Movie (1984)"	["Adventure", "Action", "Sci-Fi", "Thriller"]	4
5	"Transformers: The Last Knight (2017)"	["Adventure", "Action", "Sci-Fi", "Thriller"]	4
6	"Robotropolis (2011)"	["Adventure", "Action", "Sci-Fi", "Thriller"]	4

Figure 9: Movie with the most similar tags to "Spider-Man"

```
//simalar movie base on genre
MATCH (m:Movie)-[:GENRE]->(g:Genre)<-[:GENRE
    ]-(rec:Movie)
WHERE m.title = "Spider-Man (2002)" WITH rec
    , COLLECT(g.genresName) AS genres, COUNT
    (*) AS commonGenres RETURN rec.title,
    genres, commonGenres ORDER BY
    commonGenres DESC LIMIT 15;
```

6 CONCLUSION

This report have done a survey of neo4j. In terms of neo4j system , we have shown how data is presented by nodes and relationships and how they are stored locally. We also study the Cypher query to fetch data from the database and the protocol to send data between clients and servers. From clustering, we have shown each role of servers, Raft protocol and the life cycle of Neo4j from starting to shut down. Distributed Features are also illustrated in terms of consistency and fault tolerance. Finally, we have shown our statistics analysis results fetched by loading data into Neo4j database and using Cypher query to get data.

REFERENCES

[1] [n.d.]. *Advanced Causal Clusterin.* https://neo4j.com/docs/operations-manual/current/clustering-advanced/

[2] [n.d.]. *Clustering Introduction.* https://neo4j.com/docs/operations-manual/current/clustering/introduction/

[3] [n.d.]. *MovieLens.* https://grouplens.org/datasets/movielens/

[4] [n.d.]. *Understanding Neo4j's data on disk.* https://neo4j.com/developer/kb/understanding-data-on-disk/

[5] Faming Gong, Yuhui Ma, Wenjuan Gong, Xiaoran Li, Chantao Li, and Xiangbing Yuan. 2018. Neo4j graph database realizes efficient storage performance of oilfield ontology. *PloS one* 13, 11 (2018), e0207595.

[6] Florian Holzschuher and René Peinl. 2013. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. 195–204.

[7] Zan Huang, Wingyan Chung, Thian-Huat Ong, and Hsinchun Chen. 2002. A graph-based recommender system for digital library. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*. 65–73.

[8] Inc Neo Technology. [n.d.]. *Bolt Protocol.* https://boltprotocol.org/