How does the performance of a graph database such as Neo4j compare to the performance of a relational database such as Postgres?

Tharun Sikhinam
University of Washington
Seattle, Washington
tharun@uw.edu

1 INTRODUCTION

Relational databases have been the default choice for data storage and transaction processing applications in enterprises for a long time. In the current landscape, other database technologies are taking up a larger slice of the database market. Choosing the right database for an application is a challenging task, and this project aims to solve that challenge by comparing a relational database to a graph database.

Neo4j is chosen as the Graph database and Postgres as the relational database. The data model used to test the capabilities of these databases is the TPC-H benchmark. The TPC-H is a decision support benchmark. It consists of a suite of business-oriented ad-hoc queries and the data populating the database have been chosen to have broad industry-wide relevance. A subset of the TPC-H queries is used to compare the performance, with a focus on join heavy queries.

AWS is used to setup Neo4j and Postgres databases on EC2 instances with similar system specifications. 1GB and 10GB datasets of TPC-H data are generated to test the queries against and the query times will be reported against the database and instance type.

2 EVALUATED SYSTEM(S)

2.1 Relational Database (Postgres)

PostgreSQL is a powerful, open-source object-relational database system. It is one of the oldest relational databases and runs on all major operating systems with support for a variety of programming languages and data types. Postgres stores data in a relational model and uses SQL as its query language. Like all relational databases, the data is stored in tables in the form of rows and columns. Foreign keys are used to represent relationships between different tables. Postgres is not natively distributed and needs to be scaled up to handle more data volume. It is widely used as an OLTP database and is ACID compliant.

2.2 Graph Database (Neo4j)

Graph databases are a new type of NoSQL database that stores the data in the form of a graph model. In such databases, the nodes of a graph depict the entities while the relationships depict the association between these nodes. "Unlike other databases, graph databases store relationships and connections as first-class entities." This is the most distinguishing feature of graph databases compared to relational databases. This also translates to better performance in join related queries, and the purpose of this project is to test this claim. Neo4j is chosen as the graph database since it is one of the most widely adopted graph databases in the industry. It uses a Property Graph (PG) Model and Cypher as the query language. Similar to Postgres, Neo4j is not natively distributed. Neo4j is also ACID compliant which makes it an ideal database to compare Postgres against.

2.3 Data Model

TPC-H is a decision support benchmark. It consists of a suite of business-oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. The data model mimics a real-world OLTP application and the schema for the relational model is shown below. TPC-H dbgen tool is used to generate 1GB and 10GB datasets.

3 PROBLEM STATEMENT

How does the performance of a graph database such as Neo4j compare to the performance of a relational database such as Postgres?

The performance of both of these databases are compared in the following 5 categories

(1) Data Ingestion
(2) Size on disk
(3) Memory-constrained environment
(4) Queries
(5) Different data sizes

4 METHODOLOGY

4.1 Allocate Resources

AWS EC2 instances are used to set up the databases. Care is taken to maintain the same specifications and database configurations for both databases. A t2.micro instance (1GB RAM) is used to load the 1GB data. This test checks how the databases compare under a memory-constrained environment. t2.2xlarge ec2 instance with 32gb ram is used to compare the actual query times with the 1GB and 10gb datasets.

4.2 Data Ingestion

4.2.1 Postgres: There are many guides available that show how to load data TPC-H data onto Postgres. The first step in the process is to create the tables. A shell script is written to load the data into Postgres tables. The query ingestion times are recorded for 1 and

10GB datasets. Queries are run against this database instance using SQL.

4.2.2 Neo4j: To migrate TPC-H data model from relational to graph model, the following guide is used

(1) Table to Node Label
(2) Row to Node
(3) Column to Node Property
(4) Constraints/Indexes remain the same
(5) Foreign keys to Relationships
(6) Clean up any duplicates
(7) Join tables to relationships

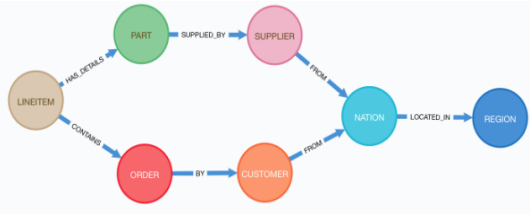

Figure 1: TPC-H graph schema

We have 8 entities in the TPCH data model but we create only 7 node labels (Supplier, Customer, Order, Nation, Region, Lineitem, and Part). Partsupp entity is replaced by a relationship. There are a total of 6 relationship types (Contains, CreatedBy, From, Locatedin, Hasdetails, and SuppliedBy) that represent all the relationships between the different entities in the TPCH model. Hasdetails relationship replaces the Partsupp entity in the TPC-H model and availqty, supplycost are stored as relationship attributes.

4.3 Query

A subset of TPC-H queries is chosen with a focus on join heavy queries. The queries are described below

(1) Large table scan TPCH-Query 6
(2) Group by, projection TPCH-Query 1
(3) 3 tables join TPCH-Query 3
(4) 4 tables join TPCH-Query 10
(5) 5 tables join TPCH-Query 7
(6) 6 tables join TPCH-Query 5
(7) 7 tables join TPCH-Query 8

SQL queries are converted to their corresponding Cypher queries and run against 1GB and 10GB datasets. The query times are reported along with data load times for both databases.

5 RESULTS

Postgres is marked in blue color and Neo4j in Maroon color in the graphs. The time along the Y-axis is in seconds.

5.1 Data Ingestion

Data Size	Postgres	Neo4j
1GB	<2 minutes	>20 minutes
10GB	<15 minutes	>3 hours

Table 1: Data ingestion times for 1GB and 10GB datasets

5.2 Size on disk

Data Size	Postgres	Neo4j
1GB	1.3GB	12.3GB
10GB	12.1GB	114.2GB

Table 2: Size on disk for 1GB and 10GB datasets

5.3 Memory-constrained environment

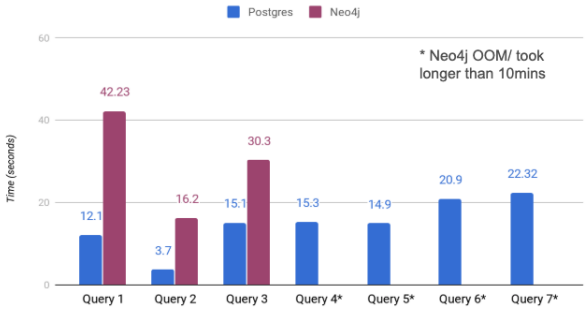

Figure 2: 1GB dataset and 1GB RAM

5.4 Query times for 1GB dataset 32GB RAM

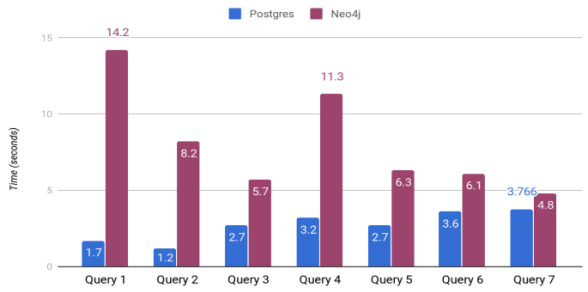

Figure 3: 1GB dataset and 1GB RAM

5.5 Query times for 10GB dataset 32GB RAM

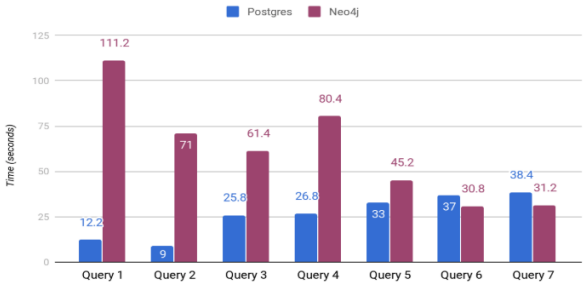

Figure 4: 1GB dataset and 1GB RAM

6 CONCLUSION

(1) **Data Ingestion:** Loading data was much easier in Postgres compared to Neo4j. Neo4j takes a lot more time in creating relationships and bulk load csv in Neo4j is not performant for large datasets.
(2) **Size on disk:** Neo4j increased the raw data size to almost 10x, with 1GB of data growing to 10GB. This increase can be attributed to the relationship information graph database stores along with nodes. Postgres had less than 1.5x increase in raw data to data stored on disk.
(3) **Memory-constrained environment:** With 1GB RAM Neo4j ran out of memory for 4 out of the 7 queries. Postgres performs much better with limited resources.
(4) **Queries:** Postgres outperformed Neo4j in almost every query under various settings. Neo4j performance benefits were observed only with joins involving more than 5 tables and for larger datasets. Large single table scans in Neo4j are very expensive. Adding indices to properties in Neo4j drastically speeds up performance. This can be a part of future work.
(5) **Data Size:** As the data size increased 10x, Postgres and Neo4j query times also increased at approximately the same rate. Neo4j seemed to perform better with larger data table joins.

Postgres performs better against Neo4j in all of the above categories. The performance benefits of Neo4j are noticed only for large datasets and joins involving more than 5 tables. These performance benefits were not significant enough to Postgres, to consider replacing relational databases in an enterprise. Overall, I would prefer to use graph databases for problems such as community detection, pathing analysis and problems involving deep recursion and continue to use relational databases for enterprise applications