Project Statement for Milestone 5

Airline Search Engine

CPTS-415_BMW

Brian Joo, Noah Waxman, Mitchell Kolb

Report Topics:

The report should cover the following subtopics and answer the questions listed:

1. Hardware and Software Platform:
   a. What hardware (computers with CPU, memory and disk storage) did you use to test scalability of the solution?

      For testing the process of scaling up or vertical scaling which is when we increase the amount of resources of individual components we used two virtual machines for this. We had our neo4j database running on a virtual machine that is setup with 1 CPU core and 4 GB of ram, and another virtual machine with 4 CPU cores and 10GB of ram. We ran both of our built-in queries from milestone 4 and recorded the execution response times to compare the difference in hardware.

      Query 1:

      ```
      MATCH (a:Airport)
      RETURN a.country AS Country, count(a) AS NumberOfAirports
      ORDER BY NumberOfAirports DESC
      LIMIT 1
      ```

      Query 2:

      ```
      MATCH (flight:Route)-[:DEPARTS_FROM]->(sourceAirport:Airport),
      (flight)-[:ARRIVES_AT]->(destinationAirport:Airport)
      WITH sourceAirport.city AS city, COUNT(flight) AS numberOfRoutes
      RETURN city, numberOfRoutes
      ORDER BY numberOfRoutes DESC
      LIMIT 5
      ```

      On the setup with 1 CPU core and 4 GB of ram.

- Query 1 executed streaming 1 record after 1ms and completed after 32ms
- Query 2 executed streaming 5 records after 16ms and completed after 284ms

On the setup with 4 CPU cores and 10GB of ram.

- Query 1 executed streaming 1 record after 1ms and completed after 26ms
- Query 2 executed streaming 5 records after 10ms and completed after 211ms

Through our testing we have concluded that neo4j uses a set amount of resources to get its systems setup and all additional resources are available to be used to process requests. On both virtual machines our requests that we developed in milestone 4 were able to be run within acceptable timings but after analyzing the results of execution times between the two we found out that for more complex queries it is more beneficial to have more resources but the benefits are negligible. Scaling up is something that our project could use if our feature set was more resource intensive but it doesn't seem like a good fit given our current situation.

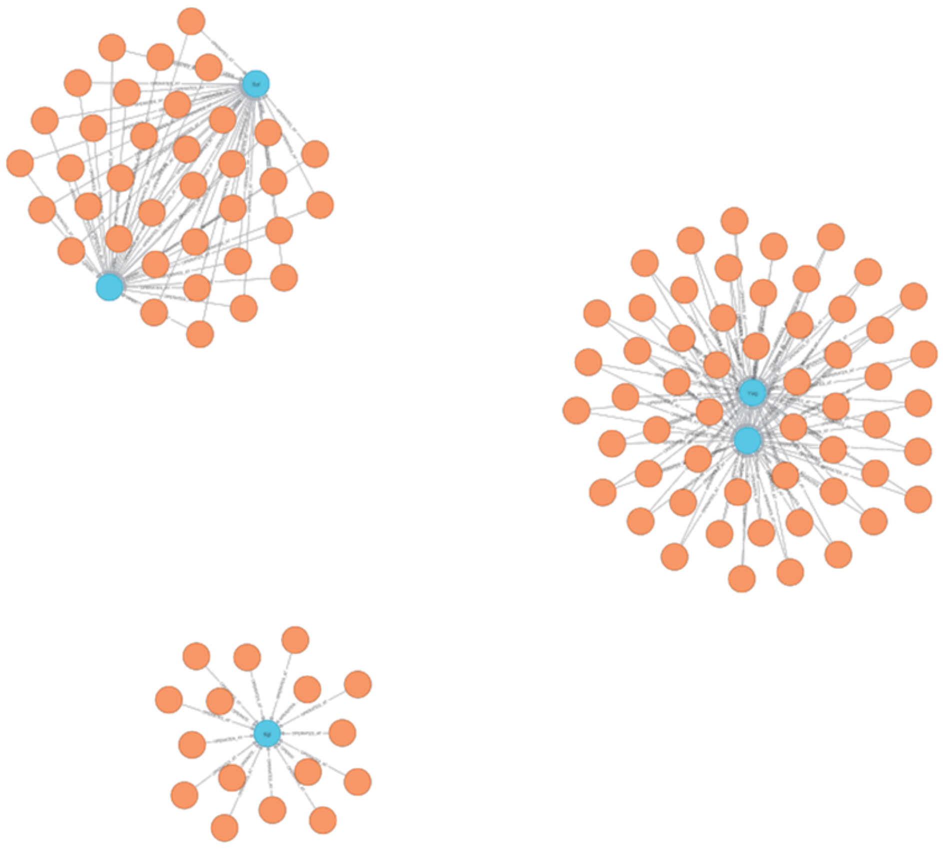
b. What software platforms (NoSQL database, distributed files, Hadoop, Spark, etc.) did you use to test the scalability of the solution?

The team was able to efficiently leverage a combination of resources that were recommended to tackle this task and to facilitate our data analysis. We used Neo4J and apache spark, as a backup. Neo4J played a critical role in managing and querying the graph data. This allowed the team to have flexibility to establish specific relationships between nodes. We tested the stability by comparing the outputs between Ne04j and apache spark by running commands such as val duration = {

```
    val t1 = System.nanoTime
    df.filter($"time zone" === "Asia/Tokyo").show(false)
    (System.nanoTime - t1) / 1e9d
  }
```

The team was also able to record the time it took to get this data by using some of the built-in tools(nantTime) to compare performance.

Neo4j is a graph database management system the team used to show some of the relationships each node had. This is important as the data would be shown visually. For example,



The team was also able to record the time it took to query certain data using some of the built in tools.

By using both Neo4J and PySpark, the team was able to obtain data that can be shown in 2 very different ways. The combination also allowed the team to test scalability. This is critical as if the data grows, the performance has to still be at a certain margin.

2. Architecture and Performance:

   a. Describe any updates you performed to your original data model or schema to better scale with data.

      There are a few improvements that we can make to our neo4j database schema to better suit scaling in the future. We looked into implementing normalization and indexing.

      For normalization we looked at our NoSQL database schema that we designed and found out that we can use more labels and relationships to create a more normalized structure of our data. We can do this by storing supplementary information from airports, airlines, countries, and planes in their own nodes. This will allow each node to have fewer properties and more meaningful relationships. This is important because as the data scales we now have the ability to only call the nodes we want and this allows there to be less processing in a distributed scaled up environment. Below is one example of our previous schema and the new normalization for the airline data.

Old:

```
LOAD CSV WITH HEADERS FROM 'file:///airlines.csv' AS row

CREATE (:Airline {airlineID: row.`airline ID`, name: row.name, iata: row.iata, icao: row.icao, country: row.country, active: row.active});
```

New:

```
LOAD CSV WITH HEADERS FROM 'file:///airlines.csv' AS row

CREATE (:Airline {airlineID: row.`airline ID`, name: row.name, country: row.country, active: row.active})

CREATE (:IATA {code: row.iata})

CREATE (:ICAO {code: row.icao})

CREATE (a:Airline {airlineID: row.`airline ID`})-[:HAS_IATA]->(i:IATA {code: row.iata})

CREATE (a)-[:HAS_ICAO]->(i:ICAO {code: row.icao});
```

For indexing we looked at our NoSQL database schema that we designed and determined that we can improve the performance of our queries if we note which queries we use frequently and create indexes for them. One of the queries we use often is finding the routes of a specific airline. Therefore we created an index on the "airline" property of the "route" nodes.

b. Describe any updates you performed to the algorithms to run them on a distributed / parallel system?

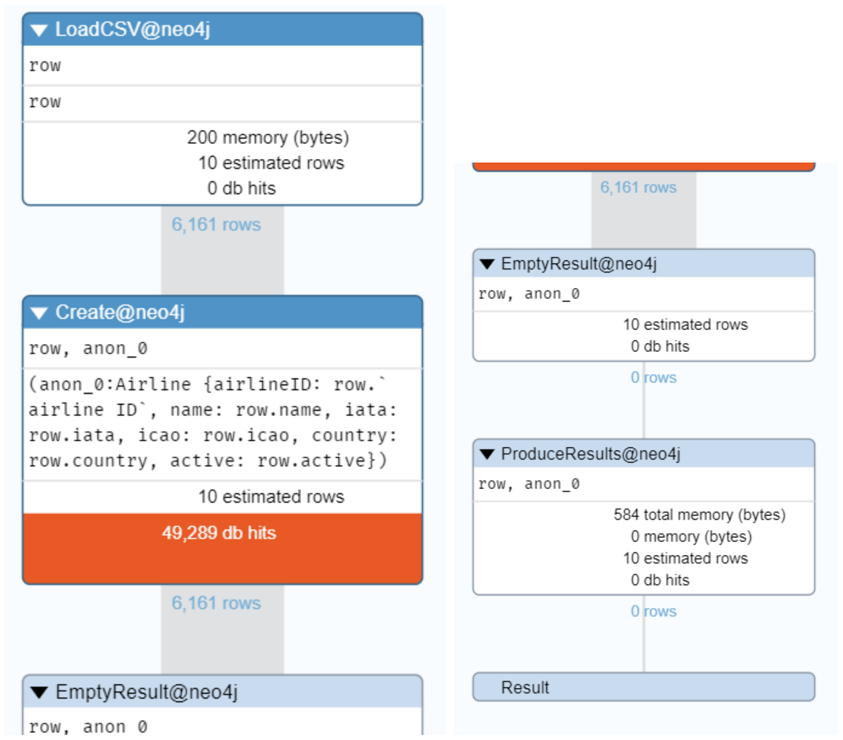
For running queries/custom algorithms in neo4j we use a language called cypher. Cypher has built in support for parallel query execution. To execute a query in parallel we would use the "CALL apoc.periodic.iterate()" function from an imported library APOC. Though since we are only able to test on local machines without cluster-mode enabled the only benefit we can get from parallelization is the ability to use other cpu threads to process queries.

c. Describe performance benchmarks on data ingestion, data query and algorithm execution?

For data injection here is benchmarks:

The performance results of the data ingestion is able to be viewed in neo4j using the cypher command “PROFILE” or “EXPLAIN”. Here is the result of the first load and create statement.

```
PROFILE LOAD CSV WITH HEADERS FROM
'file:///airlines.csv' AS row
CREATE (:Airline {airlineID: row.`airline ID`, name: row.name,
iata: row.iata, icao: row.icao, country: row.country, active:
row.active});
```



```
LoadCSV@neo4j
row
row
200 memory (bytes)
10 estimated rows
0 db hits
        6,161 rows
Create@neo4j
row, anon_0
(anon_0:Airline {airlineID: row.`
airline ID`, name: row.name, iata:
row.iata, icao: row.icao, country:
row.country, active: row.active})
10 estimated rows
49,289 db hits
        6,161 rows
EmptyResult@neo4j
row, anon_0

        6,161 rows
EmptyResult@neo4j
row, anon_0
10 estimated rows
0 db hits
        0 rows
ProduceResults@neo4j
row, anon_0
584 total memory (bytes)
0 memory (bytes)
10 estimated rows
0 db hits
        0 rows
Result
```

When ingesting the data from the airlines.csv file there are 6161 line items. To perform this command neo4j used about 584 bytes of memory. Our full dataset won’t even reach a 20th of that maximum size limit. So that means we have plenty of space to use before any we have to worry about potentially using lots of resources.

Benchmarks for built-in queries:

Query 1:

```
MATCH (a:Airport)
RETURN a.country AS Country, count(a) AS NumberOfAirports
```

```
ORDER BY NumberOfAirports DESC
LIMIT 1
```

Query 2:

```
MATCH
(flight:Route)-[:DEPARTS_FROM]->(sourceAirport:Airport),
(flight)-[:ARRIVES_AT]->(destinationAirport:Airport)
WITH sourceAirport.city AS city, COUNT(flight) AS
numberOfRoutes
RETURN city, numberOfRoutes
ORDER BY numberOfRoutes DESC
LIMIT 5
```

Executed on a setup with 4 CPU cores and 10GB of ram.

- Query 1 executed streaming 1 record after 1ms and completed after 26ms
- Query 2 executed streaming 5 records after 10ms and completed after 211ms

Benchmarks for custom algorithms:

D-Bound reachability algo (BFS):

Input: bfs(“Spokane”, 2)

Execution time: 0.028134 seconds

Shortest path algo:

Input: (“3D Aviation USA”, “Abelag Aviation”)

Execution time: 0.074112 seconds

3. User Interface and Data Visualization:
    a. Describe your updated plan for interactive user interface and data visualization, especially with larger data size.

Our original plan as stated in milestone 4 was to provide a console-based menu to the user, where they can select different options to search and retrieve information from the database. After reviewing the response from the milestone 4 we are directed to go to a more graphical UI. Our new plan for an interactive user interface is to use PySimpleGUI. This python GUI library is based off of tkinter and allows us to use 40+ widgets to build our UI. For example we can have a menu list that includes options like finding the list of airports operating in a country, finding airlines with a certain number of stops, finding active airlines in the United States, and more. The user can interact with the system by selecting a menu option and entering any additional required inputs.

As the database grows in size, we can maintain the performance of the search engine by using Neo4j's built-in indexing and query optimization features. Specifically for distributed/parallel systems, sharding would help distribute our data across multiple servers, allowing for improved availability, horizontal scalability, and better resource utilization. Even with big datasets, these features enable us to efficiently run complex queries. In addition, we make use of Cypher's query language to create efficient queries that reduce the volume of data we need to handle.

Regarding data visualization, in our case, the data we are working with - airlines, airports, routes, etc. - is largely categorical and relational. While it's possible to create visualizations such as bar charts or pie charts to represent certain aspects of the data (e.g., the number of airlines in each country), these visualizations may not provide much additional insight beyond what can be obtained from the raw data or tabular representations. For completeness though we can include a graph of our database nodes/edges using Matplotlib".

Furthermore, our data involves a lot of relationships and connections between entities - for example, routes connect airports, and airlines operate routes. While it's possible to represent these connections visually using graphs or network diagrams, such visualizations can quickly become cluttered and hard to interpret as the size of the data increases. We can reduce our dataset further, however, this would lead to inaccuracies to the data we process and the output to the user. For example, finding the most efficient route between two different airports may lead to an inaccurate/non-existent path without the rest of the data.

Therefore, in our case, we believe that a text-based interface that allows users to query and retrieve information directly is more suitable than attempting to visualize the data. This approach allows users to get the exact information they need.

4. Source Code:
    a. Provide source code of your updated data ingestion, data query and analytics algorithms in a Zip file.

Peer Evaluation:

All team members should complete the CATME Peer Evaluation survey.

Grading:

- 15 pts: Team has successfully scaled its data model, schema and algorithms for distributed / parallel computing environment. Team has a good plan on implementing an interacting user interface with visualization.

- 5 pts: Project milestone document provides all information with relevant data model, pseudo code / code snippets and diagrams / images.