Database Management Systems (CSE2004) Project

Implementing item – based Collaborative Recommendation using Graph Database

Project made by: -

1. Abhijeet Ambadekar 16BCE1156

2. K. Shashi Sekhar 16BCE1286

3. Arshad 16BCE1275

Index

1. INTRODUCTION

There has been an overload of information in the various aspects of life, such as vehicular data, information dealing with the different parts of institutions like libraries, museums, restaurants etc. It is very tough for a consumer such as a song buff who visits a song library or downloads any music app which provide him/her with a gigantic variety of content to choose from. This leaves the customer with a billions of possibilities if the shop or the app does execute a good filtering of the data based on the customer choices and ratings.

The increasing size of Movies or Songs databases have created a crucial demand for services like recommender systems in both electronic shops or libraries and also in both web and mobile applications. This enables the retailers to have better relations with clients by giving a better end-user value. This is where Graph databases, a virtually fresh concept, can provide noteworthy solutions to improve applications that have to manipulate huge amounts of data with larger levels of connections.

In graphs, a set of nodes represent entities and edges between the nodes denote the relationships between them. This provides two major advantages: first this increases the flexibility of the databases so they can handle entities with varying types and amounts of data, and secondly they increase the scalability of the database i.e. the database can handle increasing amount of continuous incoming data to be stored and processed. These properties allow graph databases to accomplish the processing, storage and manipulation of the overloading data that has been observed in this Internet-centric world.

Another notable reason to use graph databases is they provide a very insightful design of how the data is virtually spaced in the physical and conceptual scheme. One of the most famous graph database interpreters, Neo4j, is a somewhat fresh and under-developed technology but still has been able to show some impressive results in data mangling and data processing.

2. Understanding collaborative filtering

Filters systems work on two basic but very important principles:

2.1 Content based filters

Also known cognitive filtering, content based filters recommend data based on a comparison between the content of the items and a user profile. The content of each item is represented as a set of descriptors or terms, typically the words that occur in a document. The profiles of the users with same such attributes or terms are built up by studying the contents of the items viewed by them.

The major issues faced during execution of a content-based filtering systems are:

1. Terms can either be assigned automatically or manually. When terms are assigned automatically a method has to be chosen that can extract these terms from items.

2. The terms have to be represented such that both the user profile and the items can be compared in a meaningful way.

3. A learning algorithm has to be chosen that is able to learn the user profile based on seen items and can make recommendations based on this user profile.

Text documents are the most used information sources for content-based filtering systems. This provides selecting single keywords from documents as the best approach for term-analyzing. The vector space model and latent semantic indexing are two methods that use these terms to represent documents as vectors in a multi-dimensional space.

Neural networks, Relevance feedbacks, and Bayesian classifier are some the very effective learning methods for understanding and examining a user profile. Both of the vector space model and latent semantic indexing can be well used by these learning techniques to characterize forms and data-sheets.

2.2 Item based filters

These follow a more of a probabilistic style of identifying and predicting items for a user based on the ratings or evaluations on other items by him/her. It also analyses data on items bought or used by similar users to give recommendations to the users.

Rating of items is a very effective practice to implement this technique so as to find patterns that can be used to find relations between users, thus converting the existing data model into a segmentation model. User preferences are studied and the user belonging to the same data block or segment are made similar recommendations, thus building up their global profile by the help from the rating from users belonging to similar or dissimilar data segments.

These rating need not be only the ones explicitly given by the users like stars or points to a movie, book or a song. They can also be implicitly read from users, for example, the number of pages read or bookmarks made in a book of a library or number of views of a video or even as simple as songs popular from the recent locations of users.

The major issues faced by item-based filtering are:

1. Segmenting users based on their similar rating to a data item is based on as assumption that they have similar preferences which may not be the case always.

2. Segmentation of users can also neglect the similar preferences of user placed in different segments based on the ratings of some other data item(s).

3. The most important is that these systems require certain amount of time to build up accurate recommendations to new users because of the absence of past behaviours thus very less profile data.

4. User participation is very important to determine the nature of a data item and how much it is popular among users.

3. Understanding Graph Theory for Data Modelling

Section 2 gave brief descriptions on the techniques to make a recommendation. Bur the fundamental question remains the same. How should the data be stored with all the connections? In both of the user-to-user system or the item-to-item system, we can find a common idea: both can utilize a common database design to relate items to each other and find connections between elements.

Graph data models can be utilized as the best possible solution to hold all such connections. Relational databases have a conventional tabular representation of data linked with the help of foreign keys. They cannot handle the complexity of relationships among singular elements effectively, even if the data is small. However, graph databases' architecture allows the user to query the database and compute operations on their relationships swiftly and precisely.

The graph model is simple to understand: it is composed of nodes and directed relationships, both of which have properties or attributes, spatially arranged in the form of key-value pairs that define the attributes attached to the nodes (or relations). This model is really intuitive to apprehend and can be used to depict several situations using the same basic template. Though it has a rather simple design, it can be used to implement more complex scenarios.

3.1 Graph Theory

All the graph databases follow the laws as mentioned in the graph theory. Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. Graphs can be used to model many types of relations and processes in physical, biological, social and information systems. Many practical problems can be represented by graphs.

3.2 Graph algorithms and theorems

Graph theory is very famous for finding efficient algorithms to solve the general problems that occur during the implementation of graphs in real world situations. The complexity of a problem is related to the resources required to compute a solution as a function of the size of the problem. Some of the algorithms used in querying the system are discussed below:

3.2.1 Warshall's Algorithm for Reachability

The Floyd–Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices. This law and its results hold for both directed and undirected graphs (if the edge is interpreted as pair of arcs in opposite directions).

The pseudocode for this algorithm can be given as:

```
begin

E := E0

for i := 1 to n do

   for j := 1 to n do

         if (E)ji = 1 then for k := 1 to n do

               (E)jk := max((E)jk,(E)ik)

         end if

   end do

end do

end
```

3.2.2 Dijkstra's Algorithm to find the lightest path

Named after the computer scientist Edsger W. Dijkstra, this an algorithm can be used to determine the shortest paths between nodes in a graph. It was conceived in 1956.

The pseudocode for Dijkstra's algorithm is given as:

```
Begin
 for i = 0 to |V| - 1
    dist[i] = INFINITY
    prev[i] = NULL
end
dist[s] = 0
while(F is missing a vertex)
   pick the vertex, v, in U with the shortest path to s
   add v to F
   for each edge of v, (v1, v2)
        if(dist[v1] + length(v1, v2) < dist[v2])
            dist[v2] = dist[v1] + length(v1, v2)
            prev[v2] = v1
            possibly update U, depending on implementation
        end if
    end for
end while
End
```

3.2.3 The Hamiltonian Circuit

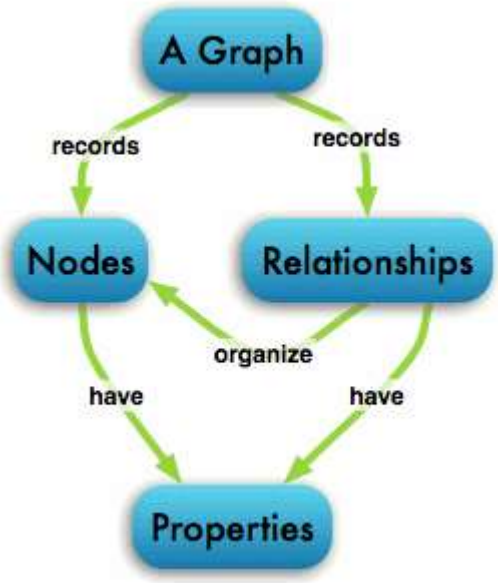
Another very popular algorithm to determine the shortest path of a cycle, if the weights of the edges have been determined. Numerous theorems have been derived from the Hamiltonian cycle.

```
bool check( path , vertex ,  posn )
 {
    for i = 1 to posn
     if path[ i ] == vertex, then
  return false
 return true
 }
void rec ( path  , posn )
 {
    if  posn  ==  v + 1 , then
         {
            print( path )
            return
         }
for i = 2 to V
    {
     path[ posn ] = i
     if (g[ path[posn-1] ][i] and check( path  , i ) )
     then rec(path ,  posn + 1 )
    }
 }
void ham_cycle()
 {
    Let path[ 0 ...V ] be an array storing the hamiltonian cycle
    for i = 1 to V
     path [i] = -1
   path[1] =  1
   rec(path , 1 )
 }
```

4. Implementation using Graph Databases

All the algorithms mentioned in the last section are just theoretical statements. One needs to have the optimum tools and technology to implement these theorems in practical situations. This section will give a brief introduction of the use of graph databases to effectively store data and rapidly query the database.

Graph database is one of the most important form of non-relational database or NoSQL database in short. They generally follow the property graph model which contains entities as nodes which can hold varying number of attributes in the form of key-value pairs. These nodes can be grouped using labels to categorize nodes having similar set of attributes. Labels can also be used to alias nodes so that they can be referenced with different keywords. Edges that connect the nodes are used to depict the relationships between nodes.



Referenced from Neo4j Home

"No broken links", this is one golden rule followed by graph databases. It means that every relationship has a starting node and an ending node, thus no node can be deleted before erasing all its accompanying relationships. So, bottom line is that every standing relationship will never have a non-existing endpoint.

5. Design and Implementation

5.1 Design

Whenever the developer has the liberty to design a NoSQL database, there are no explicit rules for its design. The main focus is to increase flexibility as there is no need to follow a rigid relational schema. That does not mean that there is no structure to the data, but certain absence of attributes or multiple linkage to a certain node is allowed. Depending on the queries that are going to be implemented in the database the design and the mindset to design it varies.

To explain the situation, a database of a fictional song recommending store is designed. It will have the following nodes and attributes:

1. 10 USER nodes with attributes Name, Gender, Fav_Genre and Fav_Artist.
2. 10 ARTIST nodes with attributes Name, Gender, Genre and Followers.
3. 21 SONG nodes with attributes Name, Length, Album, Year and Listeners.
4. 17 ALBUM nodes with attribute Name and Number_of_Likes.
5. 6 GENRE nodes with attribute Name.

These nodes will be interrelated by the following relationships:

1. SUNG_BY between SONG and ARTIST
2. IS_OF between SONG and ALBUM
3. BELONGS_TO between SONG and GENRE
4. LISTENS between USER and SONG
5. LIKES between USER and ALBUM
6. FRIEND_OF between USER and USER
7. IS_OF_TYPE between ALBUM and GENRE
8. RELEASED_BY between ALBUM and ARTIST

The overall view of the database

5.2 Implementation as a software model

CQL or Cypher Query Language is the SQL equivalent of the Graph database. Being a declarative language that provides an interface to create and query a graph database, it is designed to be human readable.

Some of the CQL commands to create the above mentioned nodes and relations are:

Command to create a User node

```
CREATE (n1:USER{NAME:'JOHN',GENDER:'M',FAV_GENRE:'POP',FAV_ARTIST:'EMINEM'})
```

Command to create a Genre node

```
CREATE(g3 : GENRE{NAME:'HIP-HOP' } )
```

Command to create a relationship between User and Song

```
CREATE (n1) - [:LISTENS] -> (s9)
```

Note that from Neo4j 2.1, the data can be loaded into the graph by using *.csv files in Cypher. This enables the user to deal with huge quantity of data.

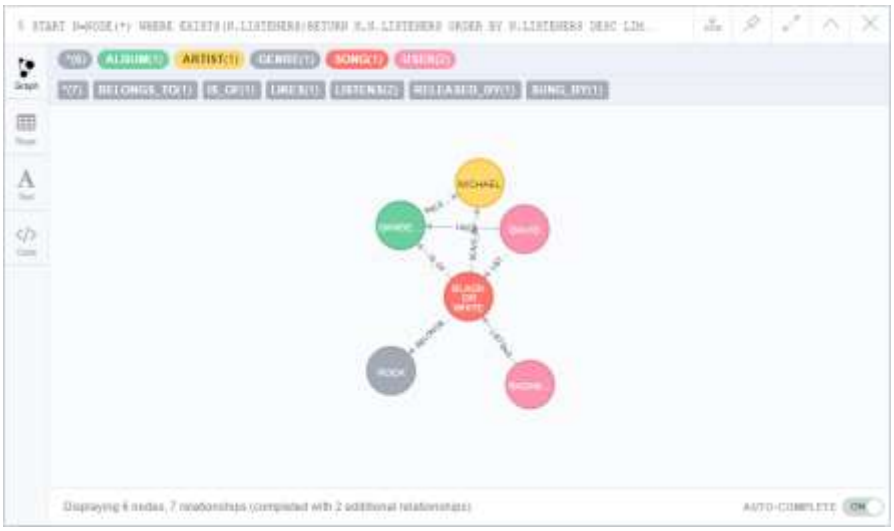
6. Querying the Database

The structuring and designing of the database has been completed. The database consists a fictional data of users having preferences on songs of particular albums belonging to specific genres. Now queries can be made to find recommendations to a specific user about any song based on a number of conditions that need to be met. For example,

To find the most popular song, the following query can be implemented:

```
START N=NODE(*) WHERE EXISTS(N.LISTENERS) RETURN N, N.LISTENERS ORDER BY N.LISTENERS DESC LIMIT 1
```

Implementing this query in the GDBMS server, the following output is received.



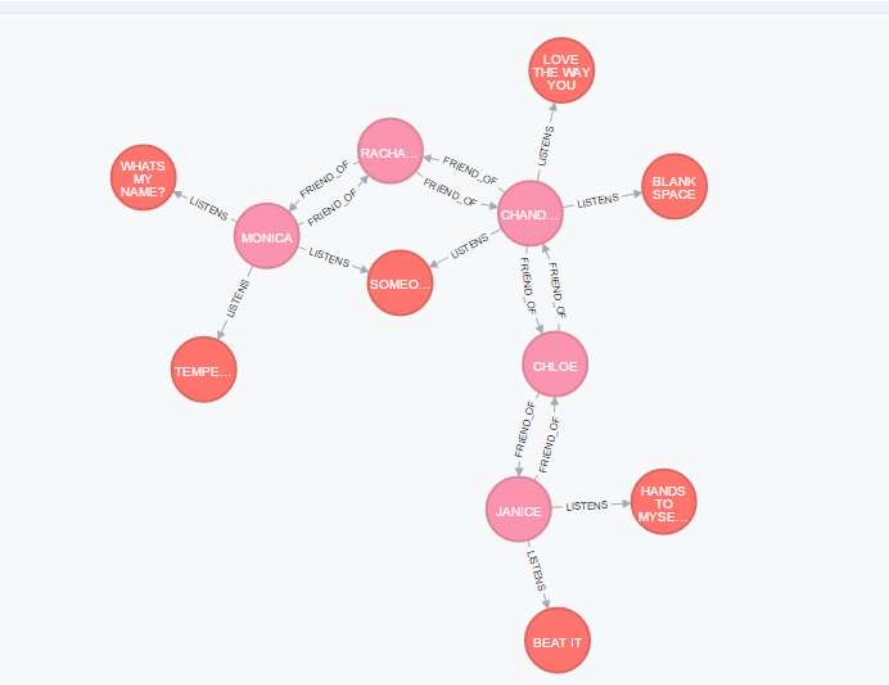
The same result in tabular format.

This query works on the principal of physical database indexing where those nodes are first found out those have the relationship as listeners and all such nodes are arranged in the decreasing order of the number of listeners and the top node of this list is returned.

Another example for a query could be recommending a User named 'Chandler' songs that are listened by friends of his friends.

```
$ MATCH (U1:USER)-[:FRIEND_OF]->(U2:USER)-[:FRIEND_OF]->(U3:USER)-[:LISTENS]->(S:SONG)WHERE
  U1.NAME='CHANDLER' RETURN U1,U2,U3,S
```

The output computed by this query is:



It can be interpreted from this query that the User 'Chandler' has only one friend of a friend 'Monica' who listens to the Songs 'What's my name' and 'Temperature', which are recommended to 'Chandler'.

7. Conclusion

Though Neo4j has given promising results for the implemented songs recommendation database, it is still under development. The graph algorithms like Floyd-Warshall's Algorithm and spanning tree algorithm along with well-defined data structures can be used to create databases that can handle vast amount of data and can find solutions to complex queries that will be executed on the data.

Since the test was conducted on a small amount of data, the evaluations on the time required to execute the queries was not same as it would be in real time situations. This sample data would be further extended to get as accurate results as possible and development is being done to implement the same on a distributed system. If everything goes well, these systems will be able to merge with the existing relational databases that are implemented in our day-to-day lives.