STUDIA INFORMATICA 2012
Volume 33 Number 2A (105)

Łukasz WARCHAŁ
Silesian University of Technology, Institute of Informatics

USING NEO4J GRAPH DATABASE IN SOCIAL NETWORK ANALYSIS

Summary. This article describes how Neo4j database capabilities can be utilized to implement measures often used in social network analysis. It gives a brief overview of the concept of Neo4j graph database. The UML class diagram of domain model is presented and discussed in details. On the basis of implementation of degree centrality and local clustering coefficient measures, several Neo4j core features are presented. In the summary, some general comments on using this database as a tool in a social network analysis are provided.

Keywords: graph database, nosql, social network analysis, neo4j

WYKORZYSTANIE GRAFOWEJ BAZY DANYCH NEO4J DO ANALIZY SIECI SPOŁECZNYCH

Streszczenie. Artykuł opisuje wykorzystanie możliwości bazy danych Neo4j w implementacji algorytmów stosowanych w analizie sieci społecznych. Przedstawiono w nim koncepcję Neo4j jako grafowej bazy danych oraz omówiono na diagramie klas podstawowe pojęcia z dziedziny przedmiotowej. Na podstawie implementacji dwóch miar (stopień węzła, lokalny współczynnik klasteryzacji), często używanych w analizie sieci społecznych, pokazano kilka podstawowych cech Neo4j. W ostatniej części znajduje się podsumowanie, w którym zebrano uwagi na temat wykorzystania tej bazy danych w analizie sieci społecznych.

Słowa kluczowe: grafowa baza danych, nosql, analiza sieci społecznych, neo4j

1. Introduction

Nowadays, social networking technologies allow people to communicate and share information with each other. Many individuals manage relationships with several close friends and hundreds of other persons, respectively. These relationships make them a part of bigger structures called communities, which in turn can also be connected to each other making social networks.

1.1. Social network analysis

Social network analysis (SNA) is focused on these relationships. It tries to find the way in which individual`s interactions with others influence their behavior or decisions [1]. Instead of investigating one`s static properties, SNA rather takes into consideration how actors cooperate and exchange information.

In real world, social networks are mainly very complex. They consist of thousands of individuals and millions of interactions (relations) between them. Performing any analyze of such a big amount of data requires building a model, which on one hand simplifies many things and on the other it is still representative.

1.2. Modeling social networks

In the most common approach, a social network is modeled as a graph *G=(V,E)* [2,3], where V is a set of nodes and E is a set of edges. Edge e_{ij} connects node v_i with v_j, so E can be defined as $E = \{e_{ij} : v_i, v_j \in V, i \neq j\}$.

When modeling a social network, each node represents an individual (actor, person). Relationships between each two individuals, which can be derived from common activities or interactions, became the edges connecting (linking) two nodes. To model the relation strength (i.e. the more messages exchanged between two persons the bigger relation strength), each edge can have a weight assigned. Set W denotes then edges weights: $\forall w_{ij} \in W \; w_{ij} \geq 0$ and the whole graph is defined as *G=(V,E,W)*.

As mentioned in the previous section, the final graph can be huge – thousands of nodes and billions of edges. Performing any analyze requires it to be persisted in some data store, which then should provide quick and efficient access to nodes and edges. This can be done using either SQL or NoSQL databases. While all well known RDBMS are optimized to store structured and organized data, persisting a graph in such database is neither straightforward nor optimal solution. Considering NoSQL databases, Key-Value, Column-Family or Document databases can be used [4]. But the most intuitive solution – assuring natural modeling – is a graph database. Among others [5], Neo4j graph database [6] is emerging, a robust and high-performance graph database. In remaining part of this paper, utilizing capabilities of this solution in SNA will be discussed. Section 2 gives a brief overview of Neo4j. In section 3 implementation of several graph measures used in SNA is presented. Section 4 gives a short summary and general conclusions about performing analysis in the Neo4j environment.

2. Neo4j database overview

Neo4j is implemented in Java programming language and can be used as an embedded or server database. In the first case, the data can be accessed through Neo4j Java API, in the second one, over the REST protocol [7].
Neo4j implements the graph data model in which core parts are nodes and relations between them. For better understanding how it is architected and organized, an UML class diagram with domain model is presented on Fig. 1.

2.1. Domain model

In every graph database, nodes and relationships play the most important role. As shown in Fig. 1, in Neo4j each *Node* can have multiple *Properties*, which are simply *key-value* pairs. Possible types of *value* are Java primitives (or arrays of): `boolean, byte, short, int, long, float, double, char, String`. Each *Node* can have different set of properties. Nodes are connected to each other by relationships. Each *Relationship* has a *Start Node* and an *End Node*. It can have an explicit *Type* and *Direction*, and also a set of *Properties*. Both nodes and relationships can be indexed, which allows to access them quickly when querying data. Indexes are implemented using *Lucene Search Engine* [8,9].

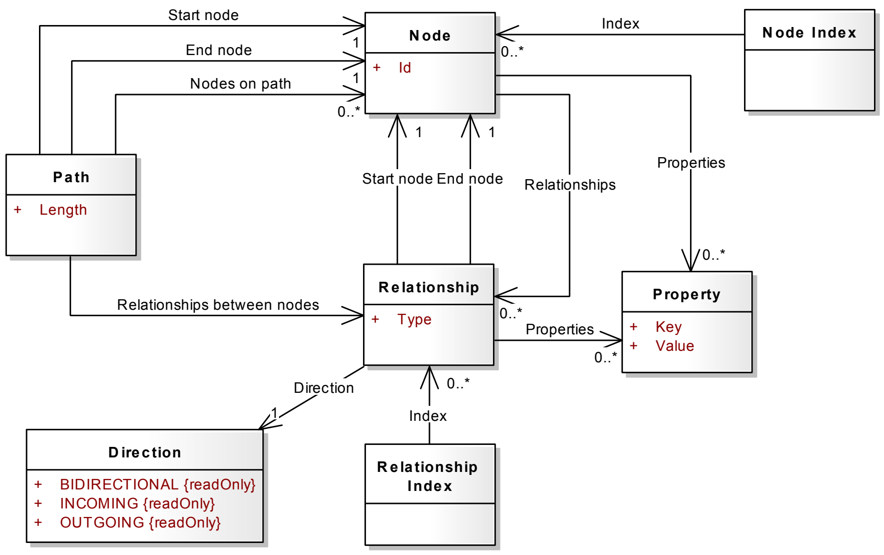

Fig. 1. Neo4j graph database domain model
Rys. 1. Model pojęć grafowej bazy danych Neo4j

Nodes connected by relationships form a *Path*. Each *Path* has beginning (*Start Node*), end (*End Node*) and *length* (number of relationships between nodes). A single *Node* creates shortest possible path with length equal to 0. *Path* is a result of visiting nodes and following their relationships according to some rules, which is called graph *traversal*.

2.2. Key features

Neo4j is a mature database with many advanced features. It supports true ACID transactions, which guarantee data consistency and reliability. It is architected in a way which allows to scale to billions of nodes and relationships. Databases can be installed on a single machine or be distributed over several machines to provide high availability. To decrease time spent on searching for particular nodes or relationships indexes can be used (also with full text search support [8]).
Neo4j does not provide explicit data encryption capabilities, but allows to use standard security solutions built into Java programming language and *Java Virtual Machine*. It is also possible to run it on an encrypted file system.

2.3. Cypher Query Language

In the Neo4j environment, data stored in a graph (nodes and relationships) can be quickly and efficiently accessed through traversals. It can be done either by direct *Traversal API* or by implemented in Neo4j *Cypher Query Language* [7]. This language is inspired by several well known and established practices for expressive querying. Many of used keywords like e.g. *WHERE, COUNT, SELECT,* have their counterparts in SQL. Construction of patterns used in filtering expressions is similar to those in SPARQL language [10]. With *Cypher*, it is possible to introduce in queries regular expressions, which are implemented using the *Scala* programming language [11].

In *Cypher*, each query can be composed of several distinct parts:

- *START* – defines start points (nodes or relationships) in the graph, pointed either by identifier or by index lookup,
- *MATCH* – defines pattern to match (graph traversal),
- *WHERE* – defines filtering criteria,
- *RETURN* – defines what should be returned as a query result.

In WHERE part it is possible to compose complex filtering expressions with boolean operators like and, or, not.

There are also several modifiers that can be applied to RETURN part to additionally shape the final result like e.g. ORDER BY to order objects in the result or LIMIT to return only subset of it.

Several example queries written in Cypher are shown in following section.

3. Performing social network analysis in the Neo4j environment

Social network analysis (SNA) is a set of methods utilized to extract knowledge from social structures. Majority of them are very complex, but frequently they use some basic measures like degree centrality, local and global clustering coefficient or other indicators based on shortest paths. While algorithm calculating shortest path between two nodes is already available in Neo4j, other two measures have to be implemented by the end user. Following sections consider possibility of implementing them using *Cypher* query language and Java code.

3.1. Datasets

Neo4j is a data store in which the social network modeled as a graph can be persisted. In this paper the scientific collaboration networks modeled as a graph are stored in Neo4j database. One of this network (graph) was created on the basis of information about publications in Institute of Informatics at Silesian University of Technology. Nodes represent authors and relationships connect those, who authored a paper together. Additionally, each relationship holds (as a property) the number of common publications. Thus the whole network is the one-mode network (with only one type of nodes) with weights on ties [12]. The second network is co-authorship network used by Newman in [17].

3.2. Degree centrality

Degree centrality is a basic, easy to calculate indicator very often used at the early stage of network studying [12]. It describes the involvement of the node in the network and can be defined as the total number of nodes connected to examined node [12,13]. In weighted networks this measure can be modified to take into consideration not only node degree but also its strength (meant as a sum of weights at relationships to other nodes). Recently, Opsahl [12] proposed degree centrality measure defined as:

$$C_D^{\omega\alpha}(i) = k_i^{(1-\alpha)} \times s_i^{\alpha} \qquad (1)$$

where k_i is the number of nodes connected to the *i-th* node, s_i is the sum of weights from relationships to other nodes and α is a positive tuning parameter. If α is between 0 and 1, high node degree is preferred, whereas if it is grater then 1 low degree is favored.

Using Neo4j as an embedded database in the Java environment this measure can be calculated using *Cypher* and little Java code. The query counting node degree and sum of weights on its ties can be write as follows:

```
start n1=node:AUTHORS_INDEX(id = '408')
match (n1)-[r:CO_AUTHOR]-(n2)
```

```
return COUNT(distinct n2) as DEGREE, SUM(r.NUM_OF_COMMON_PUBS) AS STRENGTH
```

The query first locates node *n1* with *id* property equal to 408 using index with name *AUTHORS_INDEX*. Then in *match* part, graph traversal is done – starting from node *n1*, all nodes that are connected by relationship of type *CO_AUTHOR* are located and saved in *n2* node list. As a result, query returns *n1* degree (number of distinct nodes in *n2*) and sum of weights on ties. Using this information final degree centrality can be calculated:

```
CypherParser parser = new CypherParser();
ExecutionEngine engine = new ExecutionEngine(DB_HANDLE);

Query query = parser.parse(DEGREE_QUERY_TEXT);
Map<String, Object> resultMap = engine.execute(query).iterator().next();

Integer nodeDegree = (Integer) resultMap.get("DEGREE");
Integer nodeStrength = (Integer) resultMap.get("STRENGTH");

double degreeCentrality = Math.pow(nodeDegree, 1 - ALPHA)
         * Math.pow(nodeStrength, ALPHA);
```

As shown above, using Neo4j Java API it is possible to compile text with *Cypher* query into a *Query* object, and using *ExecutionEngine* instance execute it and obtain results. Once parsed, *Query* instance can be used several times (with different parameters).

3.3. Local clustering coefficient

Among others, the degree to which nodes in a network tends to cluster together is very informative indicator, especially when analyzing real-world social networks. Many researchers find out, that in case of this kind of networks, nodes tend to cluster into a smaller groups, which are heavy interconnected inside [1416]. To observe this tendency, a global and local clustering coefficient measures were introduced [15,16]. First describes the overall level of clustering in particular network, second gives information about density of connections in node`s neighborhood.

Local clustering coefficient for the *i-th* node *v* can be defined as:

$$C_{local}(i) = \frac{\lambda_G(v)}{\tau_G(v)} \tag{2}$$

where $\lambda_G(v)$ is number of triangles on $v \in V(G)$ on graph *G*, and $\tau_G(v)$ is number of paths of length 2 centered on *v* node. In undirected graph $\tau_G(v)$ can be defined as:

$$\tau_G(v) = \frac{1}{2} k_i (k_i - 1) \tag{3}$$

where k_i is the number of nodes in *v`s* neighborhood.

In Neo4j environment local clustering coefficient for a node can be obtained using *Cypher* queries and simple Java code. According to eq. (3) denominator from eq. (2) can be easily calculated knowing node`s degree. Nominator is number of triangles that given node is part of and can be obtained by following query:

```
start n1=node:AUTHORS_INDEX(id = '408')
match p = (n1)-[:CO_AUTHOR]-()-[:CO_AUTHOR]-()-[:CO_AUTHOR]-(n1)
return COUNT(p) as NUM_OF_TRIANGLES_X_2
```

This query finds all paths that starts from node *n1* and ends on *n1* and have length 3. This corresponds to the number of triangles in *n1* neighborhood multiplied by 2, because path *n1-n2-n3-n1* and *n1-n3-n2-n1* is counted twice. Sample Java code calculating local clustering coefficient is shown below.

```
CypherParser parser = new CypherParser();
ExecutionEngine engine = new ExecutionEngine(DB_HANDLE);

Query query = parser.parse(LCC_QUERY_TEXT);
Map<String, Object> resultMap = engine.execute(query).iterator().next();

Integer triangles = (Integer) resultMap.get("NUM_OF_TRIANGLES_X_2");
if (triangles == null)
   return 0;

return triangles / (double) (degree * (degree - 1));
```

3.4. Performance

Authors claim that Neo4j is a high performance and robust database, however some advanced performance tests have to be done to prove it. Nevertheless, calculating degree centrality and local clustering coefficient measures for test datasets did not take a lot of time. Table 1 contains summary calculation time of those measures for every node in a network.

Table 1

Degree centrality and local clustering coefficient calculation time

Dataset	Nodes	Relations	Degree centrality	Local clustering coefficient
Publications in IoI at Silesian University of Technology	346	924	1.51s	9.373s
Newman's scientific collaboration network	16264	47594	10.4s	8min 55s

Each value in column 4 and 5 is an average calculation time from five runs of an experiment on a computer with 2 x 2,53GHz CPU and 4GB RAM.

4. Summary

Emerging growth of social media caused that relations between individuals became an interesting subject of scientific analysis. To perform it, interactions between people are commonly modeled as a network, and some measures like degree centrality or clustering coefficient are applied to it. Because this kind of networks is mainly very large, it is crucial to use tools that allow to calculate this measures fast. This paper shows that Neo4j database capabilities can be successfully utilized when performing analysis. The use of this database allows to naturally model a real-world network as a graph and persist it. Indexing capabilities assures that locating particular nodes is fast and easy. Implemented in Neo4j *Cypher Query Language* makes graph traversal and querying data straightforward. Nevertheless, to implement basic measures shown in this paper or those more advanced, it is necessary to write an additional code. However, features mentioned above combined with Neo4j Java API gives a solid foundation to build social network analysis tools on the top of it.

This work was partially supported by the European Union from the European Social Fund.

BIBLIOGRAPHY

1. Wasserman S., Faust K.: Social Network Analysis: Methods and Applications. Cambridge University Press, New York 1994.
2. Hanneman R. A., Riddle M.: Introduction to social network methods. University of California, Riverside CA 2005.
3. Chakrabarti D., Faloutsos Ch.: Graph Mining: Laws, Generators, and Algorithms. ACM Computing Surveys, Vol. 38, Article 2, March 2006.
4. Han J., Haihong E., Le G., Du J.: Survey on NoSQL database. In Proc. of 6th International Conference on Pervasive Computing and Applications (ICPCA), October 2011, p. 363÷366.
5. Angles R., Gutierrez C.: Survey of graph database models. ACM Computing Surv. 40, Vol. 1, Feb. 2008, p. 1÷39.
6. Neo4j Graph Database, http://neo4j.org/.
7. The Neo4j Manual, http://docs.neo4j.org/chunked/stable/.
8. Hatcher E., Gospodnetic O.: Lucene in Action. Manning Publications, 2004.
9. Apache Lucene, http://lucene.apache.org/java/docs/index.html.
10. Segaran T., Evans C., Taylor J.: Programming the Semantic Web. O'Reilly Media 2009, p. 84÷96.
11. Pollak D.: Beginning Scala. Apress, 2009.
12. Opsahl T., Agneessens F., Skvoretz J.: Node centrality in weighted networks: Generalizing degree and shortest paths. Social Networks, Vol. 32, 2010, p. 245÷251.
13. Freeman L. C.: Centrality in social networks: Conceptual clarification. Social Networks, Vol. 1, 1978, p. 215÷239.

14. Holland P. W., Leinhardt S.: Transitivity in structural models of small groups. Comparative Group Studies, Vol. 2, 1971, p. 107÷124.
15. Watts D. J., Strogatz S. H.: Collective dynamics of small-world networks. Nature, Vol. 393, 1998, p. 440÷442.
16. Opsahl T., Panzarasa P.: Clustering in weighted networks. Social Networks, Vol. 31, 2009, p. 155÷163.
17. Newman M. E. J.: The structure of scientific collaboration networks. PNAS 98, 2001, p. 404÷409.

Wpłynęło do Redakcji 31 stycznia 2012 r.

Omówienie

Artykuł omawia wykorzystanie możliwości bazy danych Neo4j w analizie sieci społecznych. W pierwszej części przedstawiono ogólną charakterystykę ww. bazy danych oraz na rys. 1 zaprezentowano diagram klas, przedstawiający pojęcia z dziedziny przedmiotowej oraz powiązania między nimi. W dalszej części zaprezentowano przykłady wykorzystania grafowej bazy danych Neo4j w analizie sieci społecznych. Jako źródło danych wykorzystano informacje o autorach i ich publikacjach w Instytucie Informatyki Politechniki Śląskiej. Następnie przedstawiono implementację dwóch podstawowych miar użytych do analizy powstałej sieci społecznej. Pierwsza z nich – stopień węzła – została wyrażona równaniem (1). Druga – lokalny współczynnik klasteryzacji – opisana została równaniem (2). Obie miary zostały zaimplementowane przy użyciu języka zapytań *Cypher* wbudowanego w Neo4j oraz języka programowania Java. Na przykładzie realizacji tych wskaźników pokazano możliwości operowania na węzłach i wykorzystywania łączących ich relacji do nawigowania po sieci (grafie).

W podsumowaniu zawarto ogólne wnioski dotyczące wykorzystania grafowej bazy danych Neo4j jako kluczowego składnika, który może zostać wykorzystany do budowy kompleksowych narzędzi do analizy sieci społecznych.

Address

Łukasz WARCHAŁ: Silesian University of Technology, Institute of Informatics,
Akademicka 16, 44-100 Gliwice, Poland, lukasz.warchal@polsl.pl.