Graph Databases

Made by:

Farah Atif (f.atif@innopolis.university)

Nikita Lozhnikov (n.lozhnikov@innopolis.ru)

Utih Amartiwi (u.amartiwi@innopolis.university)

Big Data Technology and Analysis Class 2019

CHAPTER I

INTRODUCTION

A. Background

Relational database was introduced firstly by E.F Codd in the 1970s. It was managed by Relational Database Management System (RDBMS) software. The relational database is implemented in the form of tables that are related to each other. In the mid 1980s, a standard language for managing and accessing a relational database was introduced, namely SQL (Standard Query Language). Furthermore, relational databases with RDBMS software that use the SQL language have become very popular and are widely used in database management for decades. However, relational models require a strict schema and data normalization imposed limitations on how relationships can be queried [1]. As a result, a high number of increasing data becomes a problem for relational database.

Graph theory was introduced firstly in the paper "Seven Bridges of Königsberg" written by Leonhard Euler in 1736. In mathematics and computer science, graph theory is a study of mathematical structure commonly used to model a set of objects and the relationships between these objects. Graph database is a database that uses graph structure to represent and manage the data. The flexibility of graph allows us to add entities and their relationships without affecting or changing existing data [2]. That is why many social media applications such as Facebook and Twitter use graph representations as data representations [3]. As a data scientist, it is very important for us to know how graph database works and why it can be a solution for the problem of relational database.

B. Objective

The aim of this project is to learn more about Graph database. Here we will implement it by using GraphFrames and PySpark and also set up a Neo4j database. Firstly, we will use a small dataset of papers. Here we have to find the relationship between papers. We determine a paper, then we have to find all papers that can be traced back to that paper and when there is the most papers that trace back to that paper. Then, we repeat the step for big dataset.

Furthermore, to make our understanding more deep, we will also compare the result of GraphFrames and Neo4j. Therefore, we will know how they are different and what condition that GraphFrames and Neo4j performs better.

C. Tasks Distribution

Member's Name	Tasks
Farah Atif	Implementing Graph database in Neo4j
Nikita Lozhnikov	Implementing Graph database in GraphFrame
Utih Amartiwi	Writing the report

Repository: https://github.com/farahFif/BigData-Assignment-3

CHAPTER II

BASIC OF THEORY

A. Graph Database

1. Structure and Properties of Graph Database

Graph Database is a database with graph structure. It is an online database management system with Create, Read, Update, and Delete (CRUD) methods that expose a graph data model. Graph databases are generally built for use with transactional (OLTP) systems [2]. Graph database structure:

Graph Theory	Graph Database	Represent
Vertex	Node	Entity, such as person, paper, movie, etc.
Edge	Relationship between 2 nodes	How 2 entities are associated
Attributes	Properties	Information value of node or relationship

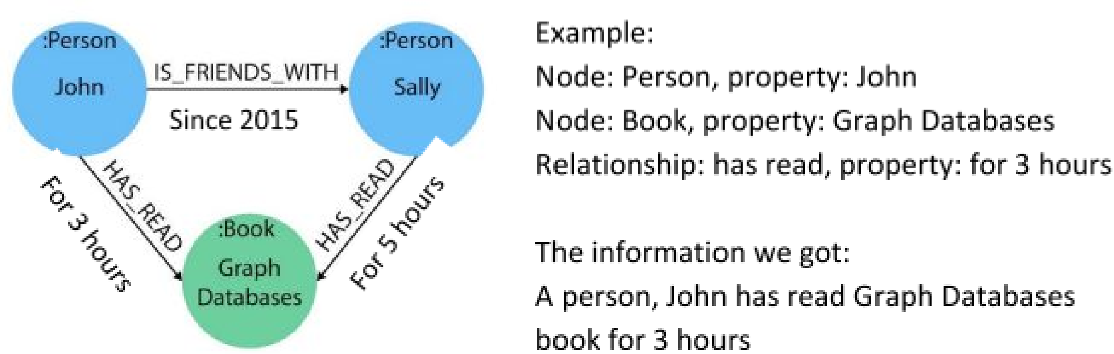


Fig. 1. Example of Graph theory implementation to Graph Database

There are two properties of graph databases:

a. Graph storage

Some graph databases use native graph storage that is optimized and designed for storing and managing graphs, while others use relational or object-oriented databases instead. Non-native storage is often slower than a native approach. [4]

b. Graph processing engine

Graph database use index-free adjacency. It means connected nodes physically "point" to each other in the database without using an index. It also called by native graph processing [4]. However, it does not mean there is no indexing in graph database. Graph database also find the pattern of the relationship and make indexing of that pattern.

2. Comparison of Relational Databases and Graph Databases

a. Performance

Relational database uses tables to represent data and its relationships. When we search the information of some related data, it will check by looking the whole table. More data we have, its performance time will be longer. Conversely, graph database has indexing pattern so that it will search only nodes that has relationship between them. As a result, the execution time will be faster.

b. Flexibility

Relational database uses fixed-schema that makes difficult to extend. Since the number of data grows significantly, relational database is less-flexible to be used. Meanwhile, graphs are naturally additive; we can add new kinds of relationships, new nodes, new labels, and new subgraphs to an existing structure without disturbing existing queries and application functionality [2].

c. Agility

As graph database is schema free, its development aligns better than relational database development with today's agile and test-driven software development practices. It is allowing graph database–backed applications to evolve in step with changing business environments [2].

B. PageRank

PageRank is an algorithm that used to measure the importance of each node in a network. It measures the number and quality of incoming relationships to a node to determine an estimation of how important that node is. In original Google paper, the PageRank formula [5]:

$$PR(u) = (1-d) + d\left(\frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_n)}{C(T_n)}\right)$$

where:

- Page u has citations from pages T_1 to $T_{n.}$
- d is a damping factor which is set between 0 and 1. It is usually set to 0.85.
- 1-d is the probability that a node is reached directly without following any relationships.
- $C(T_n)$ is defined as the out-degree of a node T.

Furthermore, now there are many ways to calculate the PageRank. In GraphFrames, there are two types of PageRank implementation. They are PageRank with a fixed number of iterations and PageRank until convergence[5]. In Neo4j, PageRank implementation has three types. First, Simple PageRank algorithm (without weight of relationship), Weighted PageRank algorithm, and Personalized PageRank [6].

C. Label Propagation

The Label Propagation algorithm (LPA) is a fast algorithm for finding clusters in a graph [5]. The steps often used for the Label Propagation are:

1) Every node is initialized with a unique label (an identifier), and, optionally preliminary “seed” labels can be used.
2) These labels propagate through the network.

3) At every propagation iteration, each node updates its label to match the one with the maximum weight, which is calculated based on the weights of neighbor nodes and their relationships. Ties are broken uniformly and randomly.
4) LPA reaches convergence when each node has the majority label of its neighbors.

CHAPTER III

RESULT ANALYSIS

A. Small Dataset

These are the papers that related to the paper *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging.*

```
$ CALL algo.pageRank.stream('Citation', 'Cites') YIELD nodeId, score RETURN algo.asNode(nodeId).title A…
```

page	score
"A Study of Three Alternative Workstation-Server Architectures for Object Oriented Database Systems"	18.65970124602318
"ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging"	9.503975622169673
"ARIES/KVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes"	5.236599185457452
"Access Support in Object Bases"	5.090104149281979
"Advanced Query Processing in Object Bases Using Access Support Relations"	5.076644926331937
"The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles"	4.664065131457515
"On Rules, Procedures, Caching and Views in Data Base Systems"	4.575133362324129
"Combination of neural and statistical algorithms for supervised classification of remote-sensing image"	4.44962190464139
"Design, Implementation and Performance Comparison of the Buddy-Tree"	4.114455318424759

Started streaming 2555 records after 54 ms and completed after 80 ms, displaying first 1000 rows.

Fig. 2. Neo4j output of some papers that related to "ARIES"

```
---- Most Influencial Papers ----

+----+--------------------+----+--------------------+--------------------+------------------+
|  id|               title|year|               venue|                desc|          pagerank|
+----+--------------------+----+--------------------+--------------------+------------------+
|1575|A Study of Three ...|1990|                VLDB|David J. DeWitt,P...|  39.1660501388246|
|1830|ARIES: A Transact...|1992|ACM Trans. Databa...|C. Mohan,Donald J...|18.870972052430776|
|1330|Access Support in...|1990|   SIGMOD Conference|Alfons Kemper,Gui...|10.490213006052292|
|2392|Advanced Query Pr...|1990|                VLDB|Alfons Kemper,Gui...|10.468468477792019|
|2094|ARIES/KVL: A Key-...|1990|                VLDB|            C. Mohan|10.399530950840836|
|1261|The R*-Tree: An E...|1990|   SIGMOD Conference|Norbert Beckmann,...| 9.166073628828169|
| 872|Combination of ne...|2000|Pattern Recogniti...|Giorgio Giacinto,...| 9.111516047705887|
|1982|On Rules, Procedu...|1990|   SIGMOD Conference|Michael Stonebrak...| 9.022156059965383|
|2353|Design, Implement...|1990|                DEXA|Bernhard Seeger,H...| 8.087885270335589|
|1861|ARIES-RRH: Restri...|1991|                ICDE|C. Mohan,Hamid Pi...| 7.781103741953357|
+----+--------------------+----+--------------------+--------------------+------------------+
```

Fig. 3. GraphFrame output of some papers that related to "ARIES"

From Figure 2 and 3 we can see that GraphFrame and Neo4j provide almost same results of paper that related to "ARIES". However, GraphFrame took longer time than Neo4j. In figure 3 we know that Neo4j only needs 80 ms from beginning till complete.

We also find which year that most papers mentioned "ARIES" book and this is the result:

```
===== Books that can be traced back
+----+-------------------+----+------------+-------------------+
|  id|              title|year|       venue|               desc|
+----+-------------------+----+------------+-------------------+
|1269|Approximate Query...|2001|      VLDB J.|Kaushik Chakrabar...|
|1640|Scalable Distribu...|2001|      RIDE-DM|Torsten Grabs,Kle...|
|2268|Efficiently Publi...|2001|      VLDB J.|Jayavel Shanmugas...|
|2276|Career-Enhancing ...|2001|SIGMOD Record|Alexandros Labrin...|
|2323|Flexible Data Cub...|2001|         ICDT|Mirek Riedewald,D...|
+----+-------------------+----+------------+-------------------+

======= The year in which the book is the most referenced ========
+----+-----+
|year|count|
+----+-----+
|1999|   74|
+----+-----+
```

GraphFrame

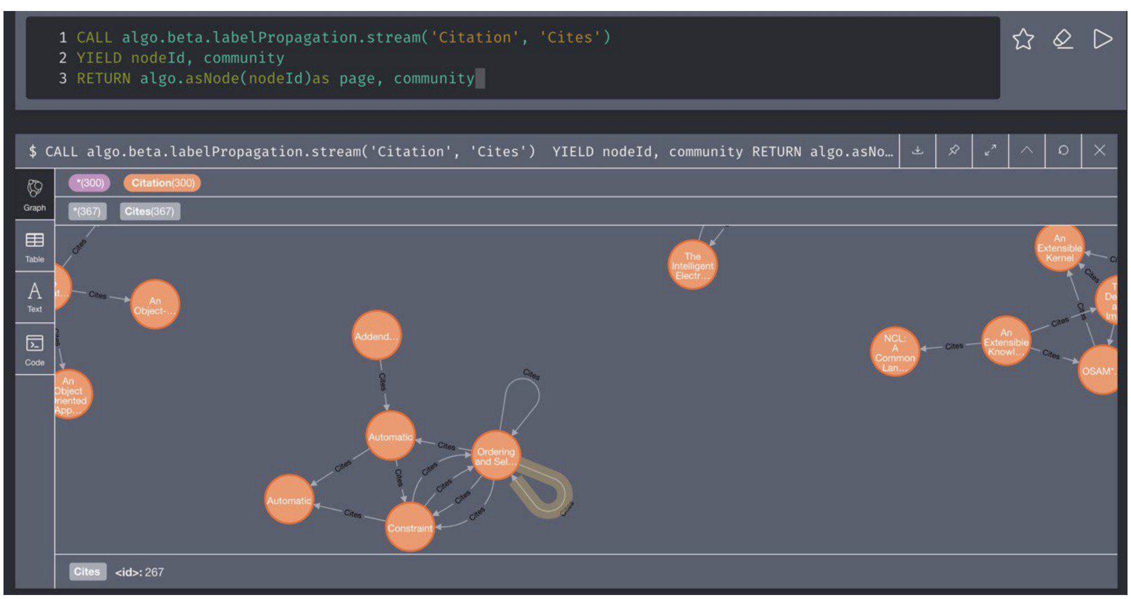
```
$ MATCH (a:Citation { title: 'ARIES: A Transaction Recovery Method Supporting Fine-(
```

counts	year
11	1993

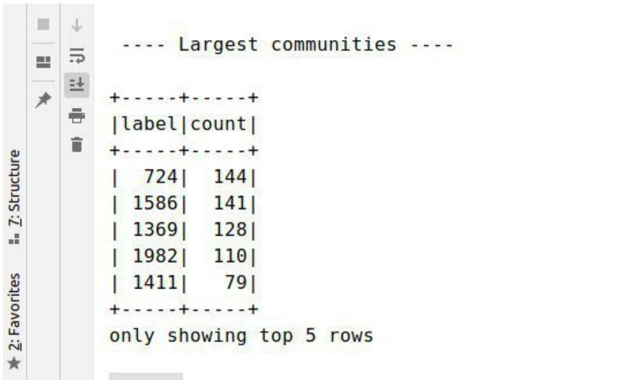
Neo4j

Fig. 4.Comparison of GraphFrame and Neo4j output of year that most paper related to “ARIES

We have different results here. GraphFrame obtain 1999 and Neo4j obtain 1993.



Neo4j



GraphFrame

Fig. 5.Comparison of GraphFrame and Neo4j output of 5 largest communities

From figure 5 we got different result representation. In GraphFrame the result is on table form and in Neo4j is on visualization form. It shows that for Neo4j can represent the result better than GraphFrame because it does not only show number but also how they are related to each other.

B. Large Dataset

For large dataset we only perform it in Neo4j because GraphFrame took too much time and need more RAM to process it. In small dataset experiment, we got that Neo4j provide better result representation and faster. So, it will be better for large dataset. These are the papers that related to the paper *Machine Learning, 2007.*

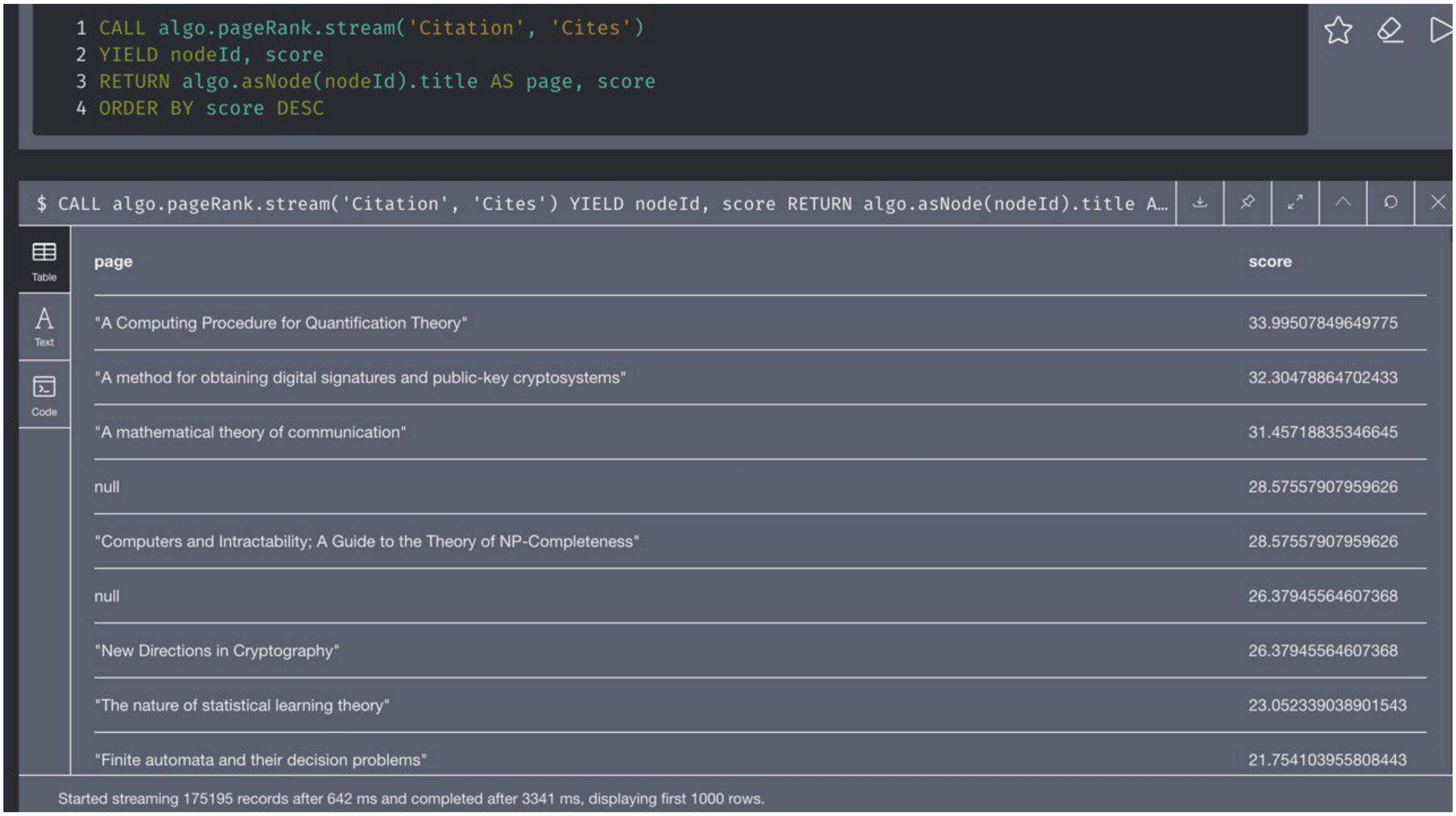


Fig. 6. Neo4j output of some papers that related to "Machine Learning, 2007"

From Figure 6 we can see that Neo4j can process and get result only 3341 ms from beginning till the end.

We also find which year that most papers mentioned "Machine Learning, 2007" book and this is the result:

Fig. 7.Neo4j output of year that most paper related to "Machine Learning, 2007"

In figure 7 we can see that most papers related to "Machine Learning" comes from year 2000.

Fig.8. Neo4j output of 5 largest communities

From figure 8 we got.

CHAPTER IV

CONCLUSION

1. Graph database is better than relational database to find the relation between the data
2. Here we have implemented graph database in GraphFrame and Neo4j. We found that in ranking 10 papers, Neo4j and GraphFrame obtain almost same result. However, Neo4j is faster than GraphFrame. Furthermore, for 5 largest communities, it provides better information than GraphFrame.

APPENDIX

Queries in Neo4j

Small Dataset

```
/Users/lozhn/Tmp/a3/v.txt
/Users/lozhn/Tmp/a3/e.txt

id, title, year, and venue information



// Delete all
MATCH (n)
DETACH DELETE n;

//
//
// SMALL
//
//

// Create Nodes

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///v.txt" AS line
FIELDTERMINATOR '\t'
CREATE (c:Citation {id: toInteger(line.id), title: line.title, year: toInteger(line.year), venue:
line.venue});


// Create Cite Index

CREATE INDEX ON :Citation(id);


// Create Relations

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///e.txt" AS line
FIELDTERMINATOR '\t'
MATCH (a:Citation),(b:Citation)
```

```
WHERE a.id = toInteger(line.from) AND b.id = toInteger(line.to)
CREATE (a)-[r:Cites]->(b);
```

In this assignment, you need to implement everything using GraphFrames and PySpark and also set up a Neo4j database and execute the same queries with Neo4j.

Read the data into a graph structure using GraphFrames (you do not need to write a parser for the small dataset. You can manually split it into two files with vertices and edges.)

Write queries that perform the following (on the small dataset):

Return all the papers that were written in 2001 and can be traced back (through citations, direct or indirect) to the paper ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. Neo4j hint. On which year, there is the most papers that trace back to the paper mentioned above? Is there any parameter of the property of the data that affect the query performance?

```
MATCH (c:Citation)-[:Cites*1..5]->(b:Citation)
WHERE c.year = 2001 AND b.title = 'ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging'
RETURN c
```

Return the most influential papers in the citation graph.

```
CALL algo.pageRank.stream('Citation', 'Cites')
YIELD nodeId, score
RETURN algo.asNode(nodeId).title AS page, score
ORDER BY score DESC
```

Discover the five largest communities. Give some description to these communities.

```
CALL algo.beta.labelPropagation.stream('Citation', 'Cites')
YIELD nodeId, community
RETURN algo.asNode(nodeId)as page, community
```

Perform the same steps on for the large dataset

When inserting new relationships, the performance can be prohibitive. Creating indexes can dramatically increase performance.

For the query on the large dataset, find all papers that can be traced back to

Machine Learning, 2007. On which year, there is the most papers that trace back to the paper mentioned above?

Be aware of missing values when processing the large data. Remember that missing values can affect joins

```
//
//
// BIG
//
//
```

Large Dataset

```
/Users/lozhn/Tmp/a3/v.txt
/Users/lozhn/Tmp/a3/e.txt

id, title, year, and venue information



// Delete all
MATCH (n)
DETACH DELETE n;

//
//
// SMALL
//
//

// Create Nodes
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///v.txt" AS line
FIELDTERMINATOR '\t'
CREATE (c:Citation {id: toInteger(line.id), title: line.title, year: toInteger(line.year), venue: line.venue});


// Create Cite Index

CREATE INDEX ON :Citation(id);


// Create Relations

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///e.txt" AS line
FIELDTERMINATOR '\t'
MATCH (a:Citation),(b:Citation)
WHERE a.id = toInteger(line.from) AND b.id = toInteger(line.to)
CREATE (a)-[r:Cites]->(b);
```

In this assignment, you need to implement everything using GraphFrames and PySpark and also set up a Neo4j database and execute the same queries with Neo4j.

Read the data into a graph structure using GraphFrames (you do not need to write a parser for the small dataset. You can manually split it into two files with vertices and edges.)

Write queries that perform the following (on the small dataset):

Return all the papers that were written in 2001 and can be traced back (through citations, direct or indirect) to the paper ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. Neo4j hint. On which year, there is the most papers that trace back to the paper mentioned above? Is there any parameter of the property of the data that affect the query performance?

```
MATCH (c:Citation)-[:Cites*1..5]->(b:Citation)
WHERE c.year = 2001 AND b.title = 'ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging'
RETURN c

MATCH (a:Citation { title: 'ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging'
```

```
})<-[:Cites]-(b:Citation)
        RETURN count(b.year) as counts, b.year as year
        ORDER BY counts DESC
```

Return the most influential papers in the citation graph.

```
CALL algo.pageRank.stream('Citation', 'Cites')
YIELD nodeId, score
RETURN algo.asNode(nodeId).title AS page, score
ORDER BY score DESC
```

Discover the five largest communities. Give some description to these communities.

```
CALL algo.beta.labelPropagation.stream('Citation', 'Cites')
YIELD nodeId, community
RETURN algo.asNode(nodeId)as page, community
```

Perform the same steps on for the large dataset

When inserting new relationships, the performance can be prohibitive. Creating indexes can dramatically increase performance.

For the query on the large dataset, find all papers that can be traced back to Machine Learning, 2007. On which year, there is the most papers that trace back to the paper mentioned above?

Be aware of missing values when processing the large data. Remember that missing values can affect joins

```
//
//
// BIG
//
//

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///paper_title.tsv" AS line
FIELDTERMINATOR '\t'
CREATE (c:Citation {id: toInteger(line.id), title: line.title});
```

```
CREATE INDEX ON :Citation(id);
CREATE INDEX ON :Citation(title);
CREATE INDEX ON :Citation(year);

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///paper_year.tsv" AS line
FIELDTERMINATOR '\t'
MERGE (c:Citation {id: toInteger(line.id), year: line.year});

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///ref.tsv" AS line
FIELDTERMINATOR '\t'
MATCH (a:Citation),(b:Citation)
WHERE a.id = toInteger(line.src) AND b.id = toInteger(line.dst)
CREATE (a)-[:Cites]->(b);


        MATCH (c:Citation)-[:Cites*1..5]->(b:Citation)
        WHERE c.year = "1997" AND b.title = "Machine Learning"
        RETURN c

        MATCH (a:Citation { title: "Machine Learning" })<-[:Cites]-(b:Citation)
        RETURN count(b.year) as counts, b.year as year
        ORDER BY counts DESC


        CALL algo.pageRank.stream('Citation', 'Cites')
        YIELD nodeId, score
        RETURN algo.asNode(nodeId).title AS page, score
        ORDER BY score DESC

        CALL algo.beta.labelPropagation.stream('Citation', 'Cites')
        YIELD nodeId, community
        RETURN algo.asNode(nodeId)as page, community
```

Query for GraphFrame

Small Dataset

```
from pyspark.sql.functions import col, lit, when
from graphframes import *
```

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql import *
from IPython.display import display
import numpy as np
from pyspark.sql.types import IntegerType
from graphframes.examples import Graphs

spark = SparkSession    \
  .builder    \
  .appName("Python Spark SQL basic example")    \
  .config("spark.some.config.option", "some-value")    \
  .getOrCreate()



vertex = spark.read.csv("file:///home/farah/Documents/v.txt",sep="	",
inferSchema="true", header="false").toDF("id","title","year","venue","desc")
edges = spark.read.csv("file:///home/farah/Documents/e.txt",sep="	",
inferSchema="true", header="false").toDF("src","dst","relationship")


g = GraphFrame(vertex, edges)

verticez = g.vertices
motifs = g.find("(a)-[e]->(b)")

pi = g.vertices.filter(" year == 2001 ")
fl = np.array(pi.select(col('id')).collect()).ravel()

motifs = motifs.filter(" b.title == 'ARIES: A Transaction Recovery Method Supporting
Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging' ")

def parcours(dejavu, graph, next):
  tovisit = []
  temp = graph.find("(a)-[e]->(b)").filter(" b.id == '"+ str(next)+"' ")
  dejavu.append(next)
  for x in (np.array(temp.select(col('e.src')).collect()).ravel()) :
    tovisit.append(x )
  return dejavu, tovisit

src = motifs.select(col('e.src')).collect()
dest = motifs.select(col('e.dst')).collect()
src = np.array(src).ravel()
```

```
dst = np.array(dest).ravel()


dejavu = []
tovisit = src
dejavu.append(dst[0])
tovi =  []
i=0
k=1
while i < k :
  if(dejavu.__contains__(tovisit[i]) == 0):
    deja , tovi = parcours(dejavu, g,tovisit[i])
    dejavu +=deja
    tovisit = np.concatenate([tovisit, np.array(tovi)])
    dejavu = list(set(dejavu))
    k = len(tovisit)
  i += 1

#print("dejavu books")
#print(dejavu)

Books = [ int(x) for x in fl if x in dejavu]
print(Books)
print(" ===== Books that can be traced back ")
verticez.filter(~verticez.id.isin(*Books) == False).show()

#================================ Year in which paper is the most referenced
============

vall = [float(x) for x in dejavu]
ids = spark.sparkContext.parallelize(vall)

row_rdd = ids.map(lambda x: Row(x))
ids = spark.createDataFrame(row_rdd,['booksid'])

print("======= The year in which the book is the most referenced ========")
final_df = verticez.join(ids,[verticez.id == ids.booksid])
counted = final_df.groupBy('year').count().orderBy(col('count'),
ascending=False).limit(1)
counted.show()

#================================ Most influencial ====================
print(" ---- Most Influencial Papers ---- \n ")
results2 = g.pageRank(resetProbability=0.15, maxIter=20)
```

```
k =results2.vertices.filter(" pagerank > 0.5 ")
k.orderBy(k.pagerank.desc()).limit(10).show()

#================================= Five largest communities
===================
print(" ---- Largest communities ---- \n")

result = g.labelPropagation(maxIter=5)

new = result.groupBy('label').count().orderBy(col('count'), ascending=False)
new.show(5)
#result.join(new, new.label == result.label).orderBy( col('count'),
ascending=False).show()
```

Large Dataset

```
from pyspark.sql.functions import col, lit, when
from graphframes import *
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql import *
from IPython.display import display
import numpy as np
from pyspark.sql.types import IntegerType

spark = SparkSession    \
  .builder    \
  .appName("Python Spark SQL basic example")    \
  .config("spark.some.config.option", "some-value")    \
  .getOrCreate()


titles = spark.read.csv("file:///home/farah/Images/large/paper_title.tsv",sep="	",
inferSchema="true", header="false").toDF("id","title")
years = spark.read.csv("file:///home/farah/Images/large/paper_year.tsv",sep="	",
inferSchema="true", header="false").toDF("id","year")

vertex = titles.join(years, titles.id == years.id).drop(years.id)

edges = spark.read.csv("file:///home/farah/Images/large/ref.tsv",sep="	",
inferSchema="true", header="false").toDF("src","dst")
```

```
g = GraphFrame(vertex, edges)

verticez = g.vertices
motifs = g.find("(a)-[e]->(b)").filter(" b.title == 'Machine Learning'")


g = GraphFrame(vertex, edges)

verticez = g.vertices
motifs = g.find("(a)-[e]->(b)")

pi = g.vertices.filter(" year == 2001 ")
fl = np.array(pi.select(col('id')).collect()).ravel()


def parcours(dejavu, graph, next):
  tovisit = []
  temp = graph.find("(a)-[e]->(b)").filter(" b.id == '"+ str(next)+"' ")
  dejavu.append(next)
  for x in (np.array(temp.select(col('e.src')).collect()).ravel()) :
    tovisit.append(x )
  return dejavu, tovisit

src = motifs.select(col('e.src')).collect()
dest = motifs.select(col('e.dst')).collect()
src = np.array(src).ravel()
dst = np.array(dest).ravel()


dejavu = []
tovisit = src
dejavu.append(dst[0])
tovi =  []
i=0
k=1
while i < k :
  if(dejavu.__contains__(tovisit[i]) == 0):
    deja , tovi = parcours(dejavu, g,tovisit[i])
    dejavu +=deja
    tovisit = np.concatenate([tovisit, np.array(tovi)])
    dejavu = list(set(dejavu))
    k = len(tovisit)
  i += 1
```

```
#print("dejavu books")
#print(dejavu)

Books = [ int(x) for x in fl if x in dejavu]
print(Books)
print(" ===== Books that can be traced back ")
verticez.filter(~verticez.id.isin(*Books) == False).show()

#================================= Year in which paper is the most referenced
============

vall = [float(x) for x in dejavu]
ids = spark.sparkContext.parallelize(vall)

row_rdd = ids.map(lambda x: Row(x))
ids = spark.createDataFrame(row_rdd,['booksid'])

final_df = verticez.join(ids,[verticez.id == ids.booksid])
counted = final_df.groupBy('year').count().orderBy(col('count'), ascending=False).limit(1)
counted.show()

#d = final_df.groupBy('year').count().orderBy(col('count'), ascending=False)
#d.agg(min('age')).show()

#================================= Most influencial ====================
print(" ---- Most Influencial Papers ---- \n ")
results2 = g.pageRank(resetProbability=0.15, maxIter=20)
k =results2.vertices.filter(" pagerank > 0.5 ")
n = k.orderBy(k.pagerank.desc()).limit(10).show()

#================================= Five largest communities ====================
print(" ---- Largest communities ---- \n")

result = g.labelPropagation(maxIter=5)

new = result.groupBy('label').count().orderBy(col('count'), ascending=False)
new.show(5)
#result.join(new, new.label == result.label).orderBy( col('count'), ascending=False).show()
```

REFERENCES

[1] G. Jaiswal and A. P. Agrawal, "Comparative analysis of Relational and Graph databases," *IOSR Journal of Engineering,* vol. 3, no. 8, pp. 25-27, 2013.

[2] I. Robinson, J. Webber and E. Eifrem, Graph Databases, Sebastopol, CA: O'Reilly Media, Inc., 2015.

[3] Z. C. Khan and T. Mashiane, "An Analysis of Facebook's Graph Search," 2014.

[4] B. Merkl Sasaki, . J. Chao and R. Howard, Graph Databases for Beginners, neo4j, 2018.

[5] M. Needham and A. E. Hodler, Graph Algorithm, Sebastopol, CA: O'Reilly Media, Inc., 2019.

[6]https://neo4j.com/docs/graph-algorithms/current/algorithms/page-rank/#algorithms-pagerank-weighted-sample