

Project: PySpark: DataFrames / SparkSQL + GraphFrames / GraphX

Student :

Presented by MANICKAM RAVISEKAR ,
Master of Science in Computer Science, 19599 , Fall Semester 2022

Professor : Dr Henry Chung
TA : Liang

SAN FRANCISCO BAY
UNIVERSITY
47671 WestingHouse Dr.,
Fremont, CA 94539

ACKNOWLEDGEMENT

One of our master's degree Project for PySpark: DataFrames / SparkSQL + GraphFrames / GraphX,

Is an Interesting, which made me to learn new things, it is useful in designing and applying using Scala, PySpark: DataFrames / SparkSQL + GraphFrames / GraphX.

For deploying this project, I would like to thank Dr. Henry Chang an TA Liang for providing all the required input .

Also, for all I would like to always pray to Almighty for giving us wisdom and power to understand things.

Content

Index :

Abstract

Pyspark / Graphx configuration

Graph Frames

Graph relationship on family and friends network

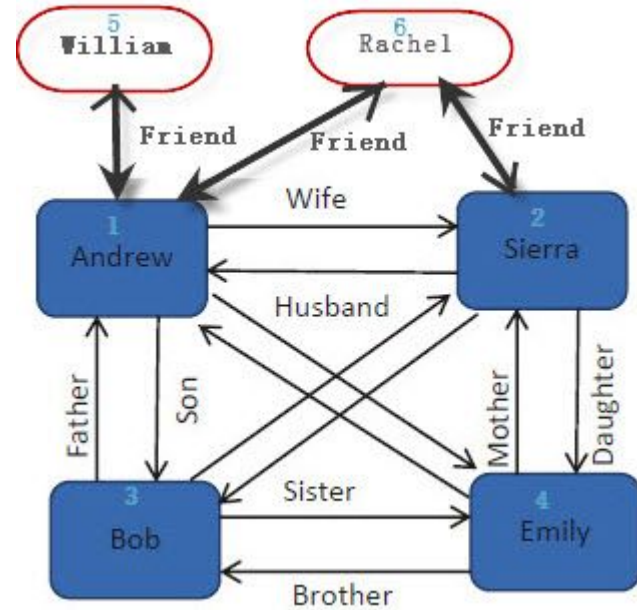
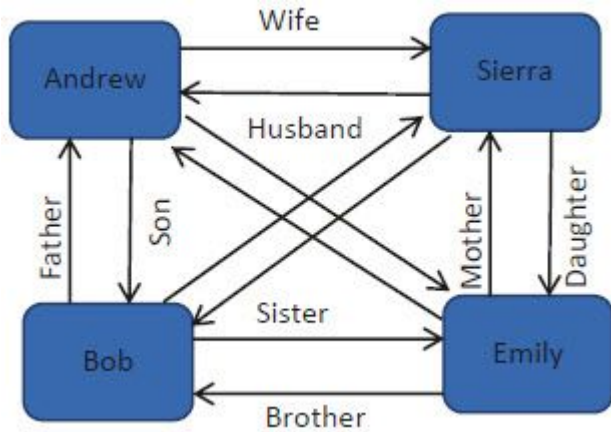
Conclusion

References

Abstract

Graph Frames are an abstraction of Data Frames that are used to do Graph Analytics. Graph Analytics stems from the mathematical Graph Theory. Graph Theory is an especially important theory used to be relationships between entities, which we can use to perform various analyses. You are using Graph Theory in your everyday life when using Google. Google introduced the PageRank algorithm that is based on Graph Theory. It tries to show the most influential website that suits your search in the best way

Dataset For Family and Friend's Relationship network



pyspark --packages graphframes:graphframes:0.8.2-spark3.2-s_2.12

hduser@cs570bigdata: ~/ggraph

```
hduser@cs570bigdata:~/ggraph$ pyspark --packages graphframes:graphframes:0.8.2-spark3.2-s_2.12
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/opt/spark/jars/spark-unsafe_2.12-3.2.3.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
:: loading settings :: url = jar:file:/opt/spark/jars/ivy-2.5.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /home/hduser/.ivy2/cache
The jars for the packages stored in: /home/hduser/.ivy2/jars
graphframes#graphframes added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-d3372d76-920b-464d-9967-3b3a5f22dd6a;1.0
  confs: [default]
  found graphframes#graphframes:0.8.2-spark3.2-s_2.12 in spark-packages
  found org.slf4j#slf4j-api:1.7.16 in central
:: resolution report :: resolve 675ms :: artifacts dl 5ms
  :: modules in use:
  graphframes#graphframes:0.8.2-spark3.2-s_2.12 from spark-packages in [default]
  org.slf4j#slf4j-api:1.7.16 from central in [default]
-----
|               |          modules          || artifacts |
|               | number| search|dwnlded|evicted|| number|dwnlded|
-----
|               | 2    | 0    | 0    | 0    || 2    | 0    |
-----
:: retrieving :: org.apache.spark#spark-submit-parent-d3372d76-920b-464d-9967-3b3a5f22dd6a
  confs: [default]
  0 artifacts copied, 2 already retrieved (0kB/30ms)
22/12/12 23:00:01 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

  ____
 /  __ \
/   /  \
/_____/

version 3.2.3

Using Python version 3.8.10 (default, Nov 14 2022 12:59:47)
Spark context Web UI available at http://cs570bigdata:4040
Spark context available as 'sc' (master = local[*], app id = local-1670866206167).
SparkSession available as 'spark'.
>>> █
```

```
#####  
# Create GraphFrames  
#####  
  
#####  
#   person dataframe : id, Name, age  
#####
```

```
hduser@cs570bigdata: ~/ggraph  
SparkSession available as 'spark'.  
>>> from graphframes import *  
>>> personsDf = spark.read.csv('/home/hduser/ggraph/persons.csv',header=True, inferSchema=True)  
>>> personsDf.createOrReplaceTempView("persons")  
>>> spark.sql("select * from persons").show()  
+---+-----+---+  
| id|   Name|Age|  
+---+-----+---+  
|  1| Andrew| 45|  
|  2|  Sierra| 43|  
|  3|    Bob| 12|  
|  4|  Emily| 10|  
|  5|William| 35|  
|  6| Rachel| 32|  
+---+-----+---+  
  
>>> █
```

relationship dataframe : src, dst, relation

relationshipDf = spark.read.csv('/home/hduser/ggraph/relationship.csv',header=True, inferSchema=True)

[relationshipDf](#).createOrReplaceTempView("relationship")

spark.sql("select * from relationship").show()

hduser@cs570bigdata: ~/ggraph

```
>>> relationshipDf = spark.read.csv('/home/hduser/ggraph/relationship.csv',header=True, inferSchema=True)
```

```
>>> relationshipDf.createOrReplaceTempView("relationship")
```

```
>>> spark.sql("select * from relationship").show()
```

```
+---+---+-----+
```

```
|src|dst|relation|
```

```
+---+---+-----+
```

```
| 1| 2| Husband|
```

```
| 1| 3|  Father|
```

```
| 1| 4|  Father|
```

```
| 1| 5| Friend|
```

```
| 1| 6| Friend|
```

```
| 2| 1|  Wife|
```

```
| 2| 3| Mother|
```

```
| 2| 4| Mother|
```

```
| 2| 6| Mother|
```

```
| 2| 6| Friend|
```

```
| 3| 1|  Son|
```

```
| 3| 2|  Son|
```

```
| 4| 1| Daughter|
```

```
| 4| 2| Daughter|
```

```
| 5| 1| Friend|
```

```
| 6| 1| Friend|
```

```
| 6| 2| Friend|
```

```
+---+---+-----+
```


- Create a GraphFrame from both person and relationship dataframes

>>> graph

GraphFrame(v:[id: int, Name: string ... 1 more field], e:[src:
int, dst: int ... 1 more field])

- A GraphFrame that contains v and e.

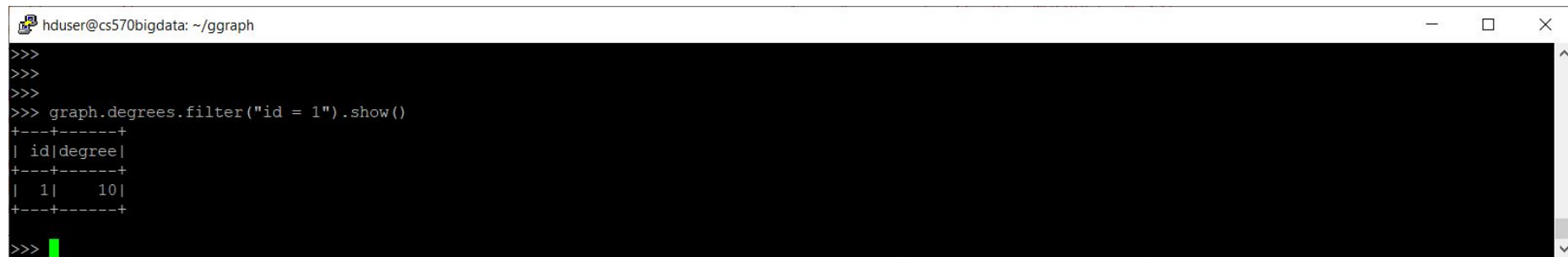
+ The v represents vertices and e represents edges.

graph = GraphFrame([personsDf](#), [relationshipDf](#))

A terminal window with a title bar showing 'hduser@cs570bigdata: ~/ggraph'. The terminal has a black background with white text. It shows the command '>>> graph = GraphFrame(personsDf, relationshipDf)' being entered, followed by a second '>>>' prompt and a green cursor. A vertical scrollbar is visible on the right side of the terminal window.

```
hduser@cs570bigdata: ~/ggraph  
  
>>> graph = GraphFrame(personsDf, relationshipDf)  
>>> 
```

```
# - Degrees represent the number of edges that are connected to a vertex.
# + GraphFrame supports inDegrees and outDegrees.
#   - inDegrees give you the number of incoming links to a vertex.
#   - outDegrees give the number of outgoing edges from a node.
# - Find all the edges connected to Andrew.
#   +--+-----+
#   |id|degree|
#   +--+-----+
#   | 1|   10|
#   +--+-----+
graph.degrees.filter("id = 1").show()
```



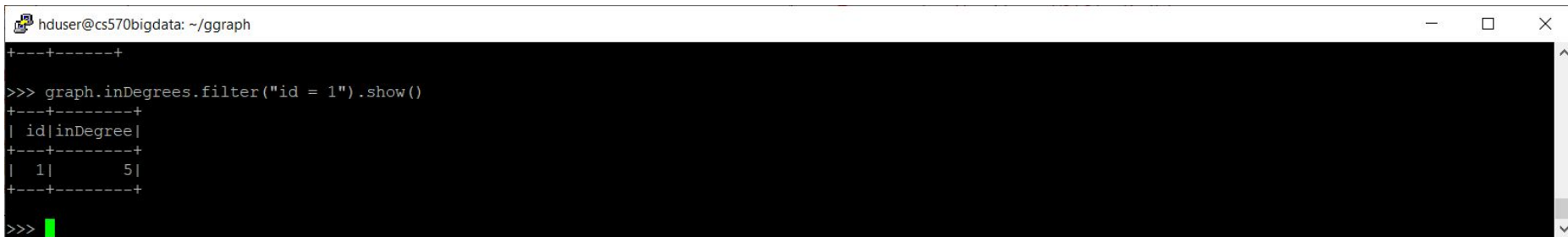
A terminal window titled 'hduser@cs570bigdata: ~/ggraph' showing the execution of the command `graph.degrees.filter("id = 1").show()`. The output is a table with two columns: 'id' and 'degree'. The table contains one row with the value 1 in the 'id' column and 10 in the 'degree' column. The terminal window has a black background and a green cursor at the end of the last prompt line.

```
>>>
>>>
>>> graph.degrees.filter("id = 1").show()
+--+-----+
| id|degree|
+--+-----+
|  1|   10|
+--+-----+
>>>
```

Find the number of incoming links to Andrew

```
# +--+-----+  
# |id|inDegree|  
# +--+-----+  
# | 1|      5|  
# +--+-----+
```

`graph.inDegrees.filter("id = 1").show()`



A terminal window titled "hduser@cs570bigdata: ~/ggraph" displays the execution of a Spark SQL query. The query is `>>> graph.inDegrees.filter("id = 1").show()`. The output is a table with two columns: `id` and `inDegree`. The first row shows `1` for `id` and `5` for `inDegree`. The terminal has a black background with white text. A green cursor is visible at the bottom left of the terminal window.

```
hduser@cs570bigdata: ~/ggraph  
+--+-----+  
  
>>> graph.inDegrees.filter("id = 1").show()  
+--+-----+  
| id|inDegree|  
+--+-----+  
|  1|        5|  
+--+-----+  
  
>>> █
```

Find the number of links coming out from Andrew using the outDegrees

```
# +--+-----+
# |id|outDegree|
# +--+-----+
# | 1|      5|
# +--+-----+
```

`graph.outDegrees.filter("id = 1").show()`



A terminal window titled "hduser@cs570bigdata: ~/ggraph" displays the execution of a Spark SQL query. The query is `>>> graph.outDegrees.filter("id = 1").show()`. The output is a table with two columns: "id" and "outDegree". The first row shows "1" and "5". The terminal has a black background with white text. A green cursor is visible at the bottom left.

```
hduser@cs570bigdata: ~/ggraph
+--+-----+
>>> graph.outDegrees.filter("id = 1").show()
+--+-----+
| id|outDegree|
+--+-----+
| 1|      5|
+--+-----+
>>>
```

#####

[Apply Triangle Counting in a GraphFrame](#)

#####

- Find how many triangle relationships the vertex is participating in

- A vertex is part of a triangle when it has two adjacent vertices with an

edge between them. For example, Andrew has 3 [triangle count](#).

+ Andrew - Sierra - Bob

+ Andrew - Emily - Bib

+ Andrew - Sierra - Emily

- A new column count is added in the output that represents the [triangle count](#).

+ The output shows that Andrew and Sierra have the maximum triangle counts,

since they are involved in 3 kinds of relationships

- Andrew as father, friend, and husband and Sierra as mother,

friend, and wife.

```
hduser@cs570bigdata: ~/ggraph
>>> personsTriangleCountDf = graph.triangleCount()
>>> personsTriangleCountDf.show()
+-----+---+-----+---+
|count| id|  Name|Age|
+-----+---+-----+---+
|    3|  1| Andrew| 45|
|    1|  6| Rachel| 32|
|    1|  3|   Bob| 12|
|    0|  5|William| 35|
|    1|  4|  Emily| 10|
|    3|  2| Sierra| 43|
+-----+---+-----+---+
>>>
```

```
# Create a "personsTriangleCount" SQL table from the
# personsTriangleCountDf DataFrame
personsTriangleCountDf.createOrReplaceTempView("personsTriangleCount")
```

```
hduser@cs570bigdata: ~/ggraph
>>>
>>>
>>> personsTriangleCountDf.createOrReplaceTempView("personsTriangleCount")
>>>
```

```
maxCountDf = spark.sql("select max(count) as max_count from personsTriangleCount")
maxCountDf.createOrReplaceTempView("personsMaxTriangleCount")
```

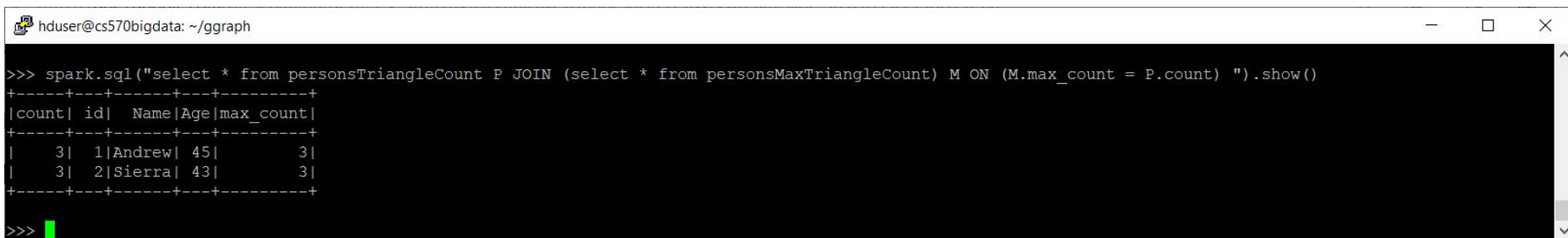
```
hduser@cs570bigdata: ~/ggraph
>>> maxCountDf = spark.sql("select max(count) as max_count from personsTriangleCount")
>>> maxCountDf.createOrReplaceTempView("personsMaxTriangleCount")
>>> maxCountDf.show()
+-----+
|max_count|
+-----+
|          3|
+-----+
>>>
```

- Join with the Persons DataFrame to be able to

view the person's details

- This is the output

```
# +----+---+-----+---+-----+
# |count| id| Name |Age|max_count|
# +----+---+-----+---+-----+
# |   3| 1|Andrew| 45|        3|
# |   3| 2|Sierra| 43|        3|
# +----+---+-----+---+-----+
```

A terminal window titled 'hduser@cs570bigdata: ~/ggraph' with standard window controls. The terminal shows a Spark SQL query being executed: 'spark.sql("select * from personsTriangleCount P JOIN (select * from personsMaxTriangleCount) M ON (M.max_count = P.count) ").show()'. The output is a table with 5 columns: count, id, Name, Age, and max_count. It contains two rows of data. The prompt '>>>' is followed by a green cursor.

```
hduser@cs570bigdata: ~/ggraph

>>> spark.sql("select * from personsTriangleCount P JOIN (select * from personsMaxTriangleCount) M ON (M.max_count = P.count) ").show()
+----+---+-----+---+-----+
|count| id|  Name |Age|max_count|
+----+---+-----+---+-----+
|   3| 1|Andrew| 45|        3|
|   3| 2|Sierra| 43|        3|
+----+---+-----+---+-----+

>>> █
```

Conclusion

Graph Theory is used in various sciences, computer science also tends to solve a lot of problems with Graph Theory. Some of the applications of Graph Theory include social media problems, travel, chip design, and many other fields.

Social media app, such as LinkedIn, which needs to connect millions of people eventually it will create enormous number of edges. To be able to apply analytics on this kind of data, regular databases will not suffice. With a regular database, we would need to apply self-joins so many times and applying self-joins will literally bring our database down so the optimal solution is graph theory application.