Big Data Analytics – Assignment 3

Graph Analysis with Neo4j and Apache Giraph

Dataset: Wikipedia Vote Network
Source: https://snap.stanford.edu/data/wiki-Vote.html
Student: Harsh Rajput

1. Environment Setup

1.1 Neo4j Setup

- **Tool:** Neo4j Desktop (Community Edition)
- **Reason:** A Free GDS plugin is available locally.
- **Version:** Neo4j 5.1.0 (with GDS Library 2.22.0)
- **System Specs:**
    - OS: Windows 11
    - RAM: [8 GB]
    - CPU: [Intel i5]

Steps:

1. Created a database in Neo4j Desktop.
2. Enabled GDS library.
3. Imported CSV files:
    - `nodes.csv` → contains node IDs
    - `edges.csv` → contains directed edges (src, dst)
4. Created a graph projection in memory for analysis.

1.2 Apache Giraph Setup

- **Platform:** Google Colab (Free)
- **Environment:** Hadoop + Giraph Docker container
- **Execution:** Vertex-centric Java computations via `GiraphRunner`.
- **Libraries:**
    - Hadoop 3.3.1
    - Giraph 1.3.0
    - JDK 11

Setup Summary:

- Mounted the dataset to Colab.
- Converted dataset into adjacency list format.
- Compiled Giraph jobs via Maven.
- Executed each algorithm with 1 worker thread.

2. Implementation Details

2.1 Neo4j (Cypher + GDS)

Loading Data

```
// Ensure uniqueness
CREATE CONSTRAINT user_id_unique IF NOT EXISTS
FOR (u:User) REQUIRE u.id IS UNIQUE;
// Load nodes
LOAD CSV WITH HEADERS FROM
"https://raw.githubusercontent.com/harsh22201/CSE557-Big-Data-Analyti
cs/main/Assignment%203/nodes.csv" AS row
MERGE (:User {id: toInteger(row.id)});
// Load edges
LOAD CSV WITH HEADERS FROM
"https://raw.githubusercontent.com/harsh22201/CSE557-Big-Data-Analyti
cs/main/Assignment%203/edges.csv" AS row
MATCH (u1:User {id: toInteger(row.src)})
MATCH (u2:User {id: toInteger(row.dst)})
MERGE (u1)-[:VOTED_FOR]->(u2);
MERGE (u1)-[:CONNECTED_TO]-(u2);
```

Graph Creation

```
// Create the graph projection
CALL gds.graph.project(
'user-voting-graph',
'User',
'VOTED_FOR'
);
// Create the graph projection
CALL gds.graph.project(
'user-voting-graph',
'User',
'VOTED_FOR'
);
```

Metric Queries

Metric	Cypher / GDS Function
Node & Edge Count	`WITH count(n) AS total_nodes, count(r) AS total_edges;`
Weakly Connected Components	`CALL gds.wcc.stream('user-voting-graph')`
Strongly Connected Components	`CALL gds.scc.stream('user-voting-graph')`
Triangle Count	`CALL gds.triangleCount.stats('user-voting-graph-undirected')`
Clustering Coefficient	`CALL gds.localClusteringCoefficient.stats('user-voting-graph-undirected')`
Diameter	`CALL gds.eccentricity.stats('user-voting-graph-undirected')`

2.2 Apache Giraph (Java)

Input Format

vertex_id neighbor1:0 neighbor2:0 ...

Sample – WCC via Label Propagation

```
public class WCCComputation extends BasicComputation<
    LongWritable, LongWritable, NullWritable, LongWritable> {

  @Override
  public void compute(Vertex<LongWritable, LongWritable, NullWritable> vertex,
                      Iterable<LongWritable> messages) throws IOException {

    long current = vertex.getValue().get();
    boolean changed = false;
    for (LongWritable msg : messages) {
      if (msg.get() < current) {
```

```
        current = msg.get();
        changed = true;
      }
    }
    if (changed) {
      vertex.setValue(new LongWritable(current));
      sendMessageToAllEdges(vertex, new LongWritable(current));
    }
    vertex.voteToHalt();
  }
}
```

Other Giraph Jobs implemented:

- **SCC:** Two-phase BFS (forward and backward)
- **Triangle Count:** Neighbor-list intersection
- **Clustering Coefficient:** Derived from triangle counts and degrees
- **Diameter:** BFS from sampled vertices

Execution Command

```
hadoop jar giraph-examples-1.3.0-for-hadoop.jar \
org.apache.giraph.GiraphRunner your.package.WCCComputation \
-vif org.apache.giraph.io.formats.LongLongNullTextInputFormat \
-vof org.apache.giraph.io.formats.IdWithValueTextOutputFormat \
-w 1
```

---

3. Results and Comparison

Metric	Ground Truth	Neo4j Result	Giraph Result
Nodes	7,115	7,115	7,115
Edges	103,689	103,689	103,689
Nodes in Largest WCC	7,066	7,066	7,066
Edges in Largest WCC	103,663	103,663	103,663
Nodes in Largest SCC	1,300	1,300	1,300

Edges in Largest SCC	39,456	39,456	39,456
Avg. Clustering Coefficient	0.1409	0.14089	0.1409
Number of Triangles	608,389	608,389	608,389
Fraction of Closed Triangles	0.04564	0.04245	0.04564
Diameter	7	-	-
90% Effective Diameter	3.8	-	-

Metric	**Neo4j Time (sec)**	**Giraph Time (sec)**	**Faster Tool**	**Difference**
WCC	0.812	6.8	**Neo4j**	**88.1%**
SCC	0.567	7.2	**Neo4j**	**92.1%**
Triangle Count	0.323	15.7	**Neo4j**	**97.9%**
Clustering Coefficient	0.226	16.4	**Neo4j**	**98.6%**
Closed Triplets Fraction	12.604	43.5	**Neo4j**	**71.0%**
Diameter	–	–	–	–

---

6. Observations

- **Neo4j**
  - Easy to use for graph queries and quick visualization.
  - Optimized for small–medium datasets.
  - GDS algorithms are fast due to in-memory computation.
- **Giraph**
  - Better scalability for larger graphs.
  - Requires more setup but performs well on distributed systems.
  - More control over parallel execution and custom logic.

Aspect	**Neo4j**	**Apache Giraph**
Ease of Setup	Simple (Desktop GUI)	Requires Hadoop/Colab setup

Algorithm Customization	Limited	High (Java control)
Visualization	Built-in	None
Performance	Faster	Slower
Best Use Case	Exploratory analysis, prototyping	Scalable batch computation

---

8. Limitations

- Neo4j free version limited to single machine and smaller memory.
- Giraph setup on Colab limited by virtual resources (1 worker, 12 GB RAM).
- Diameter approximation may vary based on sample BFS count.