

Large Scale Graph Processing

DBMS Term Project



DBMS144

Group Members

Banoth Karthik	-	21CS10013
Chappa Jayanth	-	21CS10017
Parala Siva Sai Yeswanth	-	21CS30034
Kandula Revanth	-	21CS10035
Nuthakki Varun	-	21CS10045

Objective

The objective of this project is to process large graphs in a database using a graph processing system such as ApacheGraph, Pregel (GoldenOrb), Giraph, or Stanford GPS.

The objective of processing large graphs in a database has several practical applications in various fields such as social network analysis, web analysis, bioinformatics, and cybersecurity.

Libraries/Frameworks/Tools:

- Apache Spark GraphX library
- PageRank algorithm used to compute Pagerank value
- Prigel algorithm used to compute shortest Path between vertices which is based on message passing to neighbours
- Scala Standard Library
- Apache Spark Server:- Apache Spark is a distributed computing framework that is designed for processing large-scale data sets. It provides an interface for distributed data processing that can be used to perform various operations such as batch processing, stream processing, machine learning, and graph processing.

Methodology

Stanford Network Analysis Project

- Network was collected by crawling Amazon website.
- It is based on Customers Who Bought This Item Also Bought feature of the Amazon website.
- If a product i is frequently co-purchased with product j , the graph contains a directed edge from i to j .

SCALA

- Scala is used in this code because Apache Spark, the framework being utilized, provides an API primarily designed for Scala.
- While Spark also supports other programming languages like Python and Java, Scala is often favored due to its compatibility with Spark's functional programming paradigm and concise syntax.

Methodology

Query_processing.scala:-

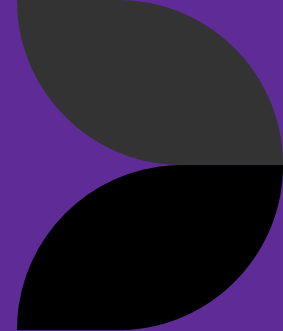
- The query_processing.scala code uses the Apache Spark GraphX library to achieve our goal.
- The code performs the steps like loading the graph using GraohLoader.edgeListFile function, providing the menu of options, performing a total of 7 graph queries based on the user's choice.
- The code finds the shortest path between two vertices, given their IDs, using the **PRAGEL ALGORITHM**, which is a message-passing algorithm for computing shortest paths in graphs.
- The code first initializes the vertex attributes to positive infinity, except for the source vertex, which is initialized to 0.0. Then, it sends messages along the edges to update the distances, and iterates until convergence, using the graph.mapVertices and graph.pregel methods of the GraphX library. Finally, the code selects the shortest path and prints it using the println statement.



Queries

```
Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 17.0.10)
Type in expressions to have them evaluated.
Type :help for more information.

scala> :load /Users/varun_nuthakki/term-project/dbms/query-processing.scala
Loading /Users/varun_nuthakki/term-project/dbms/query-processing.scala...
import org.apache.spark.graphx._
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
inputFile: String = /Users/varun_nuthakki/Desktop/amazon0302.txt/
graph: org.apache.spark.graphx.Graph[Int,Int] = org.apache.spark.graphx.impl.GraphImpl@1ad03f80
continue: Boolean = true
Choose an option:
1. Number of vertices
2. Number of edges
3. Neighbors of a vertex
4. Number of triangles in the graph
5. Top k vertices with max degree
6. Shortest path between two vertices
7. Connected components
8. Exit
Number of vertices: 262111
```



Choose an option:

1. Number of vertices
2. Number of edges
3. Neighbors of a vertex
4. Number of triangles in the graph
5. Top k vertices with max degree
6. Shortest path between two vertices
7. Connected components
8. Exit

Enter an option number: Number of edges: 1234877

Choose an option:

1. Number of vertices
2. Number of edges
3. Neighbors of a vertex
4. Number of triangles in the graph
5. Top k vertices with max degree
6. Shortest path between two vertices
7. Connected components
8. Exit

Enter an option number: Enter vertex ID: Neighbors of vertex 45: 46, 46, 58, 59, 59, 88, 88

Choose an option:

Choose an option:

1. Number of vertices
2. Number of edges
3. Neighbors of a vertex
4. Number of triangles in the graph
5. Top k vertices with max degree
6. Shortest path between two vertices
7. Connected components
8. Exit

24/04/15 01:27:25 WARN ShippableVertexPartitionOps: Joining two VertexPartitions with different indexes is slow.

24/04/15 01:27:25 WARN ShippableVertexPartitionOps: Joining two VertexPartitions with different indexes is slow.

24/04/15 01:27:25 WARN ShippableVertexPartitionOps: Joining two VertexPartitions with different indexes is slow.

24/04/15 01:27:25 WARN ShippableVertexPartitionOps: Joining two VertexPartitions with different indexes is slow.

Number of triangles in the graph: 717719

Choose an option:

1. Number of vertices
2. Number of edges
3. Neighbors of a vertex
4. Number of triangles in the graph
5. Top k vertices with max degree
6. Shortest path between two vertices
7. Connected components
8. Exit

Enter an option number: Enter k:

(14949,425)

(4429,409)

(33,366)

(10519,339)

(12771,335)

(8,298)

(297,280)

(481,280)

(5737,277)

(9106,232)

(8939,226)

(93,224)

Shortest path between 1 and 6: 2.0

Choose an option:

1. Number of vertices
2. Number of edges
3. Neighbors of a vertex
4. Number of triangles in the graph
5. Top k vertices with max degree
6. Shortest path between two vertices
7. Connected components
8. Exit

Enter an option number: Enter source vertex ID: Enter destination vertex ID: 24/04/15 01:35:27 WARN BlockManager: Block rdd_2046_1 already exists on this machine; not re-adding it

24/04/15 01:35:27 WARN BlockManager: Block rdd_2046_3 already exists on this machine; not re-adding it

24/04/15 01:35:27 WARN BlockManager: Block rdd_2094_2 already exists on this machine; not re-adding it

24/04/15 01:35:27 WARN BlockManager: Block rdd_2094_1 already exists on this machine; not re-adding it

24/04/15 01:35:28 WARN BlockManager: Block rdd_2318_1 already exists on this machine; not re-adding it

24/04/15 01:35:28 WARN BlockManager: Block rdd_2366_3 already exists on this machine; not re-adding it

24/04/15 01:35:28 WARN BlockManager: Block rdd_2414_1 already exists on this machine; not re-adding it

Shortest path between 1 and 6: 2.0

Choose an option:

```
Vertex 7007 belongs to cluster 0
Vertex 118105 belongs to cluster 0
Vertex 76754 belongs to cluster 0
Vertex 217755 belongs to cluster 0
Vertex 54961 belongs to cluster 0
Vertex 179607 belongs to cluster 0
Vertex 150556 belongs to cluster 0
Vertex 176404 belongs to cluster 0
Vertex 124708 belongs to cluster 0
Vertex 39458 belongs to cluster 0
Vertex 13610 belongs to cluster 0
Vertex 16876 belongs to cluster 0
Vertex 246151 belongs to cluster 0
Vertex 194455 belongs to cluster 0
Vertex 220303 belongs to cluster 0
Vertex 109205 belongs to cluster 0
Vertex 83357 belongs to cluster 0
Vertex 54306 belongs to cluster 0
Vertex 204800 belongs to cluster 0
Vertex 178952 belongs to cluster 0
Vertex 153104 belongs to cluster 0
Vertex 42006 belongs to cluster 0
Vertex 12955 belongs to cluster 0
Vertex 124053 belongs to cluster 0
Vertex 248699 belongs to cluster 0
Vertex 111753 belongs to cluster 0
Vertex 137601 belongs to cluster 0
Vertex 251965 belongs to cluster 0
Vertex 219648 belongs to cluster 0
Vertex 82702 belongs to cluster 0
Vertex 207348 belongs to cluster 0
Vertex 41351 belongs to cluster 0
Vertex 0 belongs to cluster 0
Choose an option:
1. Number of vertices
2. Number of edges
3. Neighbors of a vertex
4. Number of triangles in the graph
5. Top k vertices with max degree
6. Shortest path between two vertices
7. Connected components
8. Exit
Enter an option number:
```

Methodology

Pagerank.scala:-

- The pagerank.scala code uses the Apache Spark GraphX library to compute the PageRank algorithm on a graph dataset. The graph dataset is loaded from an edge list file using the GraphLoader.edgeListFile method, which creates a Graph object with vertex IDs of type Long, edge attributes of type Int, and vertex attributes of type Int.
- The PageRank algorithm is a link analysis algorithm used by search engines to rank web pages in their search results. It was developed by Larry Page and Sergey Brin, the founders of Google, and it forms the basis of Google's original search algorithm.
- The fundamental idea behind PageRank is to assign a numerical weight to each element of a hyperlinked set of documents, such as web pages, nodes of a graph with the purpose of measuring its relative importance within the set. The algorithm views links between nodes as votes of trust. A node that is linked to by many other nodes is considered more important than a node with fewer links


```

scala> :load /Users/varun_nuthakki/term-project/dbms/pagerank.scala
Loading /Users/varun_nuthakki/term-project/dbms/pagerank.scala...
import org.apache.spark.graphx.GraphLoader
import org.apache.spark.graphx.{Graph, VertexId}
import org.apache.spark.rdd.RDD
import org.apache.spark.graphx.lib.PageRank
inputFile: String = /Users/varun_nuthakki/Desktop/amazon0302.txt/
graph: org.apache.spark.graphx.Graph[Int,Int] = org.apache.spark.graphx.impl.GraphImpl@7e350e11
numIterations: Int = 10
pageRankGraph: org.apache.spark.graphx.Graph[Double,Double] = org.apache.spark.graphx.impl.GraphImpl@b5a5f98
top10: Array[(org.apache.spark.graphx.VertexId, Double)] = Array((33,303.5811108728533), (93,269.3278359583615), (8,225.82631824787137), (94,219.0
3048375620546), (2501,211.37991673953016), (4429,202.63908289102534), (56,190.22280992057975), (95,187.62095486637602), (151,182.94107816351874),
(2353,166.59436027895464))
Top 10 vertices by PageRank:
33      303.5811108728533
93      269.3278359583615
8       225.82631824787137
94      219.03048375620546
2501    211.37991673953016
4429    202.63908289102534
56      190.22280992057975
95      187.62095486637602
151     182.94107816351874
2353    166.59436027895464

scala>

```

Profile_performance:-

- The `profile_performance.scala` code is used to profile the performance of loading a graph using Apache Spark GraphX.
- The `ThreadMXBean` class is used to get information about the CPU utilization of the current thread.
- The `System.nanoTime()` function is used to get the start and end time of the graph loading process, which is used to calculate the ELAPSED TIME.
- This code can be used to profile the performance of loading large-scale graph datasets using Apache Spark GraphX. By measuring the elapsed time and CPU utilization, it is possible to optimize the performance of the graph loading process by adjusting the cluster resources, partitioning the graph data, or using more efficient algorithms for graph processing

```
scala> :load /Users/varun_nuthakki/term-project/dbms/profile_performance.scala
Loading /Users/varun_nuthakki/term-project/dbms/profile_performance.scala...
import scala.io.StdIn.readLine
import org.apache.spark.graphx.GraphLoader
import org.apache.spark.sql.Session
import java.lang.management.ManagementFactory
import java.lang.management.ThreadMXBean
import java.nio.file.{Files, Paths}
threadBean: java.lang.management.ThreadMXBean = com.sun.management.internal.HotSpotThreadImpl@6f46426d
starttime: Long = 57498786081333
startCpuTime: Long = 13399053000
graphPath: String = /Users/varun_nuthakki/Desktop/amazon0302.txt/
disksUtilized: Long = 245107195904
graph: org.apache.spark.graphx.Graph[Int,Int] = org.apache.spark.graphx.impl.GraphImpl@7237d62b
endtime: Long = 57499533775833
elapsedCpuTime: Long = 230664000
elapsed: Double = 0.7476945
cpuUtilization: Double = 30.850033001446448
The graph file is utilizing 245107195904 disk(s)
ElapsedTime for loading graph: 0.7476945
CPU utilization: 30.850033001446448%
The graph file is utilizing 245107195904 disk(s)

scala>
```

The slide features a white background with the text "Thank you" in a large, black, sans-serif font. On the left side, there is a partial view of a purple circle. On the right side, there is a vertical purple bar with a dark grey circle partially visible behind it, and another dark grey circle partially visible below it.

Thank you