DATABASE MANAGEMENT SYSTEMS

TERM PROJECT

Large scale graph processing

Abhishek Ragala - 20CS10048
Vishnu Vardhan Sailada - 20CS10051
Chenna Keshava Reddy - 20CS10014

OBJECTIVE:

In this project, we aimed to process large graphs using Spark GraphX, a graph processing library that is built on top of Apache Spark. We've loaded a dataset from Stanford's SNAP large graph repository. Our goal was to find different communities in these datasets while keeping the graph's modularity in mind. We then provided an interface to run multiple queries, along with profile performance.

Building upon the power of Spark GraphX, our objective was to embark on a thrilling journey of processing large graphs from Stanford's SNAP repository. Our primary focus was on discovering distinct communities within the graph, while also paying meticulous attention to the graph's modularity to unlock hidden patterns and insights. To make the process seamless, we designed a user-friendly interface that enables users to effortlessly run multiple queries and delve deep into the graph's intricacies. Additionally, we were determined to profile the performance of our graph processing tasks, diligently measuring various metrics such as computation time, memory utilisation, and scalability. Through innovative approaches, creative problem-solving, and a keen eye for performance optimization, our objective was to deliver a robust and efficient solution for large-scale graph processing that unlocks the full potential of Spark GraphX.

Methodology:

Data Acquisition: We obtained a large graph dataset from Stanford's SNAP large graph repository, which served as the input for our graph processing tasks. The dataset was carefully selected based on the project's objectives and requirements.

Environment Setup: We installed and configured Spark GraphX, a powerful graph processing library built on top of Apache Spark, on our designated environment. This involved setting up the necessary software and hardware components, as well as configuring Spark GraphX to optimise performance for our specific dataset.

Graph Processing Tasks: We implemented various graph processing tasks using Spark GraphX to accomplish our objectives. This included utilising graph algorithms such as path between two nodes and degree of node. We also implemented queries and computations to extract relevant information from the graph, such as PageRank calculations.

User Interface Development: We designed and developed a user-friendly interface that allowed users to interact with the graph processing system. This interface provided an intuitive way for users to run multiple queries, specify parameters, and visualise the results, making it easy for non-technical users to explore and analyse the graph data. Performance Profiling: We conducted performance profiling to evaluate the efficiency and effectiveness of our graph processing tasks. This involved measuring various performance metrics, such as computation time, memory utilisation, and scalability, to assess the system's performance and identify any potential bottlenecks or optimizations needed.

QUERIES:

- What are the total number of nodes in the graph?
- What is the total number of edges in the graph?
- What is the degree of a node?
- What are the neighbours of a node?
- Check whether there is an edge between two nodes?
- What is the Number of triangles in the graph?
- What is the Average degree in the graph?
- What are Number of connected components in the graph?
- What is PageRank of the graph?
- What is the maximum degree in the graph?
- What is Distance between two nodes?

PERFORMANCE

CPU Utilisation: We use the ThreadMXBean class from the java.lang.management package to measure CPU utilisation. It captures the CPU time consumed by the current thread before and after the execution of a particular function, and calculates the CPU utilisation as a percentage of elapsed time. This measure helps in understanding how much of the CPU's processing capacity is being utilised by the program during its execution.

Disk Utilisation: We use the java.nio.file package to calculate the total space utilised by the graph file stored on the disk. It uses the Files.getFileStore and Paths.get methods to obtain the file store associated with the graph file's path, and retrieves the total space occupied by the file store. This measure helps in understanding the amount of disk space utilised by the graph file, which can be useful in monitoring disk usage and planning for storage requirements.

Elapsed Time: We capture the start and end times of the execution of various functions using the System.nanoTime() method, which measures time in nanoseconds. It calculates the elapsed time by taking the difference between the end time and the start time, and converts it to seconds by dividing by 1e9. This measure helps in understanding the time taken by each function to complete its execution, which can be useful in identifying performance bottlenecks and optimising the code.

Memory Usage: We use the Runtime.getRuntime() method to obtain the runtime object, and calculate the memory used by the program by taking the difference between the total memory and the free memory. This measure helps in understanding the amount of memory utilised by the program during its execution, which can be useful in monitoring memory usage and optimising memory management.

RESULTS

```
import java.nio.file.{Files, Paths}
threadBean: java.lang.management.ThreadMXBean = sun.management.ThreadImpl@543588e6
starttime: Long = 23346067050745
startCpuTime: Long = 26277274602
graphPath: String = /home/ragala/spark/code/facebook_combined.txt
disksUtilized: Long = 153681051648
graph: org.apache.spark.graphx.Graph[Int,Int] = org.apache.spark.graphx.impl.GraphImpl@351157
endtime: Long = 23346465914249
elapsedCpuTime: Long = 345195784
elapsed: Double = 0.398863504
cpuUtilization: Double = 86.54484066308558
The graph file is utilizing 153681051648 disk(s)
ElapsedTime for loading graph: 0.398863504
CPU utilization: 86.54484066308558%
defined object FunctionSwitch
```

```
defined object FunctionSwitch
Enter a number to select a function, or 'q' to quit:
1. number_of_nodes
2. number_of_edges
3. Degree of a Node
4. Neighbors of a node
5. To check a edge between two nodes
6. Number of triangles :
7. Average degree
8. Connected_components
9. Pagerank :
10.Max_Degree :
11.Distance_between_nodes :
Enter the input:
```

```
Enter a number to select a function, or 'q' to quit:
1. number_of_nodes
2. number_of_edges
3. Degree of a Node
4. Neighbors of a node
5. To check a edge between two nodes
6. Number of triangles :
7. Average degree
8. Connected_components
9. Pagerank :
10.Max_Degree :
11.Distance_between_nodes :
Enter the input: 1
No of nodes are:
4039
ElapsedTime for the query: 0.058371407
CPU utilization: 15.150777503101818%
Enter a number to select a function, or 'q' to quit:
1. number_of_nodes
2. number_of_edges
3. Degree of a Node
4. Neighbors of a node
5. To check a edge between two nodes
6. Number of triangles :
7. Average degree
8. Connected_components
9. Pagerank :
10.Max_Degree :
11.Distance_between_nodes :
Enter the input: 2
No of edges :
88234
ElapsedTime for the query: 0.048399403
CPU utilization: 23.050296715436758%
Enter a number to select a function, or 'q' to quit:
1. number_of_nodes
2. number_of_edges
3. Degree of a Node
4. Neighbors of a node
5. To check a edge between two nodes
6. Number of triangles :
7. Average degree
8. Connected_components
9. Pagerank :
10.Max_Degree :
11.Distance_between_nodes :
Enter the input: q
Memory Used : 179191184 bytes
```

REFERENCES

Apache Spark official documentation:
https://spark.apache.org/docs/latest/index.html

Scala programming language official documentation:
https://docs.scala-lang.org/

GraphX documentation (Graph processing library for Apache Spark):
https://spark.apache.org/docs/latest/graphx-programming-guide.html

Java Management Extensions (JMX) official documentation (for understanding how to monitor CPU utilisation and memory usage):
https://docs.oracle.com/en/java/javase/16/management/introduction.html

Java NIO (New I/O) official documentation (for understanding how to work with file I/O in Java):
https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/nio/package-summary.html

Facebook Social Network dataset documentation (for understanding the structure and properties of the Facebook Combined dataset):
https://snap.stanford.edu/data/egonets-Facebook.html

Scala String Interpolation documentation (for understanding how to use string interpolation in Scala):
https://docs.scala-lang.org/overviews/core/string-interpolation.html

Graph algorithms and graph processing theory (for understanding various graph algorithms, graph processing techniques, and graph theory concepts):
https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms

Big data and distributed computing concepts (for understanding the fundamentals of big data, distributed computing, and related technologies):
https://en.wikipedia.org/wiki/Big_data,

Github link -
https://github.com/chenna14/-Large-scale-graph-processing

THANK YOU