Minimum Spanning Tree Using MapReduce Framework

Introduction

What is a Minimum Spanning Tree?

- A Minimum Spanning Tree (MST) is a subset of edges of a weighted undirected graph that connects all the vertices with a minimum possible total weight.
- Computing an MST is useful in Networking (for telecommunications), planning potential new facilities, etc. This makes it a fundamental problem in graph theory.

Sequential MST Algorithms

- Traditional MST algorithms include:
  - Prim's Algorithm'
  - Kruskal's Algorithm
  - Boruvka's Algorithm
- Large-scale graphs cannot fit into memory on a single machine.
- Distributed frameworks like MapReduce enable scalable processing.
- Leverages parallel computation to handle big data efficiently.

Sequential MST Algorithms

Why not Kruskal's or Prim's?

- Complexity
- Not ideal for large graphs.
- Prim's might be more efficient but still difficult.

MapReduce Framework Overview

- MapReduce is a programming model for processing large datasets.
- Two main steps:
  - **Map**: Processes input data in parallel and produces key-value pairs.
  - **Reduce**: Aggregates intermediate results to produce final output.
- Features:
  - Fault tolerance
  - Scalability
  - Parallel Processing

What is Boruvka's Algorithm?

- Named after Otakar Boruvka, who proposed it in 1926.
- One of the earliest algorithms for finding the MST.
- Works by progressively merging connected components of the graph.
- Utilizes the "greedy approach" to select the minimum-weight edges.
- Particularly efficient for parallel and distributed computing.

Key Steps of Boruvka's Algorithm

1. Start with all vertices as separate components.
2. For each component:
   - Find the minimum-weight edge connecting it to another component.
3. Add the selected edges to the MST and merge connected components.
4. Repeat until all vertices are in a single component.

MST Using MapReduce

- Approach based on Boruvka's Algorithm.
- Key Steps:
  - Find the minimum edge for each component (Map phase).
  - Merge components using selected edges (Reduce phase).
  - Repeat until all components are merged into one.

Implementation Details

- Input: Graph represented as edges with weights.
- Output: Set of edges forming the MST.
- Key Components:
  - Edge partitioning for parallel processing.
  - Component labeling and merging.
  - Iterative MapReduce steps.
- Technologies Used:
  - PySpark for distributed processing.

Results

- Generated on our own datasets.
- Performance metrics:
  - Time to compute MST (scales with graph size).

No. of Nodes	Time
100	0.17
1000	1.15
10000	19.94
100000	40.32

Table: Performance (No. of nodes in graph vs time)

Conclusion

- Implemented MST using MapReduce with PySpark.
- Achieved efficient processing of large-scale graphs.
- Future Work:
  - Explore alternative distributed frameworks.
  - Explore other algorithms and other distributed settings that might seem useful.

Thank you!