# Implementation Report: Dijkstra's Algorithm with Apache Spark on Azure

Nishchay Patel

April 28, 2025

**Abstract**

This report details the implementation and performance analysis of Dijkstra's shortest path algorithm using Apache Spark deployed on Microsoft Azure Virtual Machines (VMs). It covers the distributed algorithm design, the Azure infrastructure setup, encountered challenges, and key lessons learned during the process, focusing on both algorithmic and infrastructural aspects.

## 1 Introduction

This report describes the implementation of Dijkstra's shortest path algorithm using Apache Spark, executed within a cluster environment hosted on Azure Virtual Machines. The objective was to leverage Spark's distributed computing capabilities to efficiently compute the shortest paths from a designated source node to all other reachable nodes in a large, weighted graph.

## 2 Implementation Details

### 2.1 Algorithm Overview

Dijkstra's algorithm is a well-known greedy algorithm designed to find the shortest paths in a graph with non-negative edge weights. This implementation adapts the classic single-node algorithm for parallel execution in a distributed setting using Apache Spark's Resilient Distributed Dataset (RDD) abstraction, enabling processing across multiple compute nodes.

### 2.2 Key Components

- **Graph Representation:** The graph structure (nodes and weighted edges) is represented as an adjacency list stored within Spark RDDs, allowing for parallel processing of graph partitions.
- **Distance Tracking:** A distributed data structure, conceptually similar to a dictionary but managed across Spark executors (RDDs of key-value pairs), maintains the shortest known distance from the source to each vertex discovered so far.
- **Active Nodes Set:** An RDD maintains the set of nodes whose distances were updated in the previous iteration. This set is used to focus computation in the subsequent iteration.
- **Broadcast Variables:** The current state of shortest distances (potentially filtered or aggregated) is efficiently distributed to all worker nodes using Spark's broadcast mechanism for read-only access during distance relaxation calculations.

### 2.3 Implementation Strategy

The distributed algorithm proceeds iteratively:

1. **Initialization:** Parse the input graph text file: weighted-graph into an RDD representing the adjacency list. Initialize an RDD mapping the source node to distance 0 and all other nodes to infinity. Initialize the active nodes set containing only the source node.
2. **Iteration:** Repeat the following steps until the active nodes set is empty:

- Broadcast the current shortest distances for the active nodes.
- Use the graph RDD and the active nodes RDD to generate candidate paths and distances ('flatMap'). Each active node proposes new distances to its neighbors.
- Aggregate these candidate distances ('reduceByKey'), selecting the minimum distance for each node reached.
- Join the newly calculated minimum distances with the existing shortest distances RDD.
- Update the main distances RDD where a shorter path has been found.
- Identify the set of nodes whose distances were updated; this becomes the active nodes set for the next iteration.

3. **Termination:** The algorithm terminates when an iteration completes with no updates to any node's shortest distance, indicating convergence.

# 3 Performance Analysis

1. Source Node: 0
2. Total Iterations: 14
3. Algorithm Time: 3.211 seconds
4. Total Execution Time: 5.772 seconds
5. Read and Setup Time: 2.557 seconds
6. Reachable Nodes: 10000/10000
7. **Metric:** Execution time per iteration was recorded:

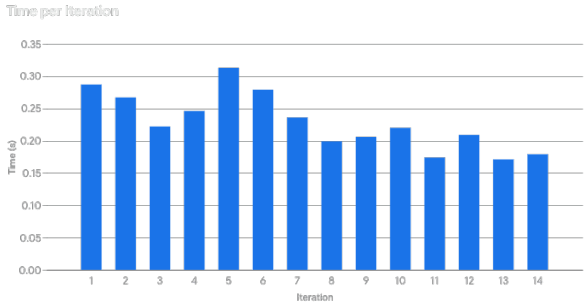


Figure 1: Time taken per iteration

# 4 Challenges and Lessons Learned

## 4.1 Challenges Encountered

- **Azure Network Configuration:** Correctly configuring Azure Network Security Group (NSG) rules to allow communication between Spark master, my machine, and the driver program was critical and initially challenging. Debugging blocked ports required careful checking of traffic flow and NSG priorities.
- **Distributed State Management:** Designing the RDD transformations to correctly and efficiently update distributed distances while minimizing data shuffling (e.g., during 'reduceByKey' or 'join' operations) remained a complex aspect of the Spark implementation.

### 4.2 Lessons Learned

- **Infrastructure is Key:** The stability and performance of the underlying Azure infrastructure (VMs, networking, storage) are as crucial as the Spark code itself. Proper VNet and NSG planning upfront saves significant debugging time.
- **Understanding Spark Execution Model:** Gained a deeper appreciation for how Spark translates RDD transformations into stages and tasks, the cost of shuffling data, and the utility of broadcast variables and accumulators in distributed computations.
- **Cloud Networking Fundamentals:** Developed practical skills in configuring Azure VNets, subnets, and NSGs, understanding the necessity of explicitly allowing ports for distributed applications like Spark.

## 5 Conclusion

This project successfully demonstrated the implementation of Dijkstra's shortest path algorithm on a distributed platform using Apache Spark hosted within Microsoft Azure. The solution effectively utilizes Spark RDDs for parallel computation and broadcast variables for efficient state sharing. Key challenges involved both the intricacies of distributed algorithm design within Spark and the practicalities of configuring the Azure network and compute environment.