Assignment 1

Aryaman Srivastava 2210110206

Implementing Vector Clocks

1. Introduction

This report presents a comparative analysis of two vector clock implementations in distributed systems:

Standard Vector Clock (VC) - The traditional approach where complete vector clocks are transmitted with every message

Singhal-Kshemkalyani (SK) Optimization - An optimized approach that transmits only relevant vector clock entries

Our experiments demonstrate that the SK optimization achieves significant reduction in message overhead, with the optimization transmitting approximately 67-74% fewer entries per message compared to the standard approach.

2. Parameters

Number of processes (n): Varied from 10 to 15 (increments of 1)

Inter-event time (λ): 5 ms

Event ratio (α): 1.5 (internal to message send events)

Messages per process (m): 50

Topology: Fully connected topology

In this topology:

- Every process is connected to every other process
- Each process i has edges to all processes j where j ≠ i
- For n processes, each process has (n-1) neighbors

For example:

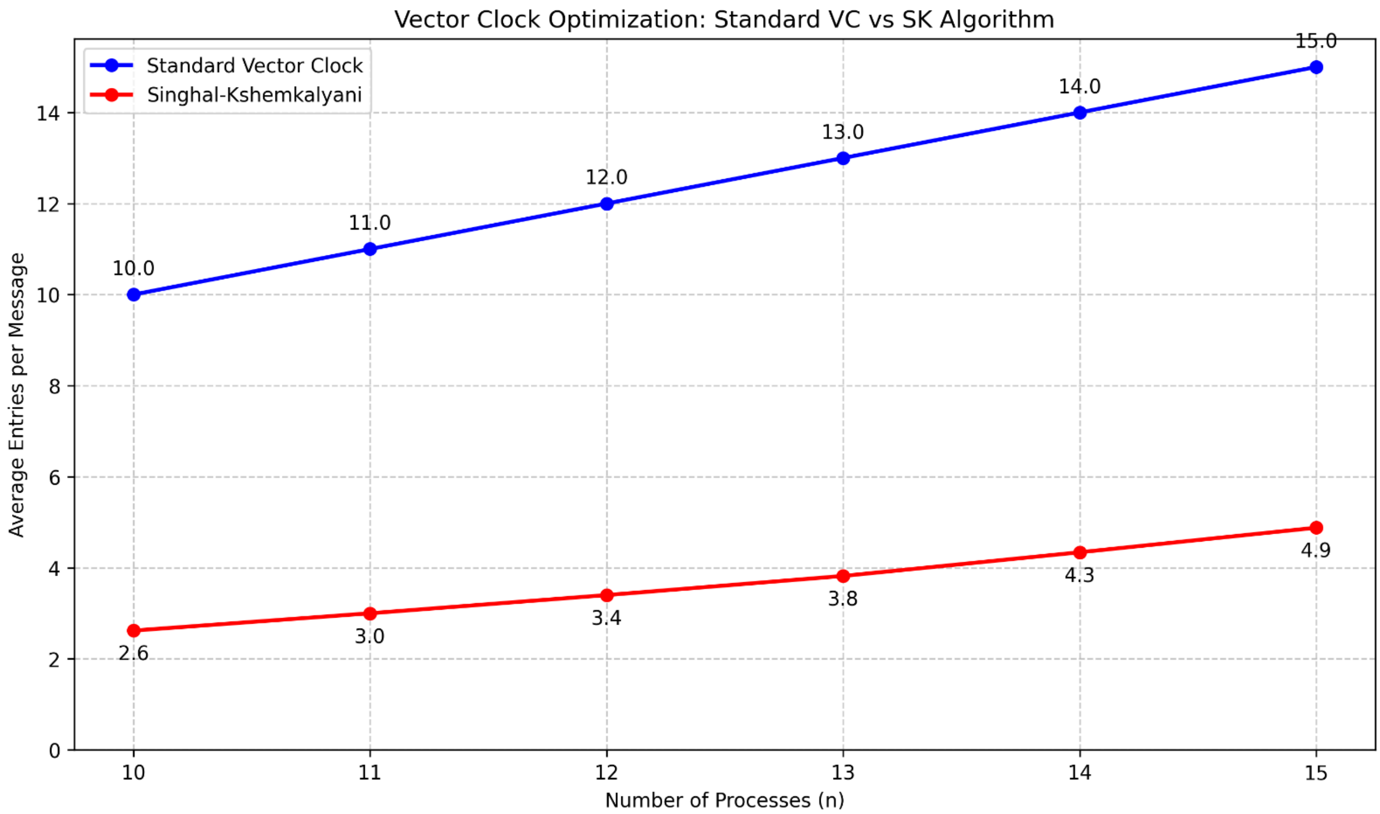
If n=3, process 1's neighbors are [2,3], process 2's neighbors are [1,3], and process 3's neighbors are [1,2]

If n=4, process 1's neighbors are [2,3,4], process 2's neighbors are [1,3,4], etc.

3. Results

Number of Processes (n)	Standard VC (entries/msg)	SK Optimization (entries/msg)	Reduction (%)
10	10.0	2.6	74.0%
11	11.0	3.0	72.7%
12	12.0	3.4	71.7%
13	13.0	3.8	70.8%
14	14.0	4.3	69.3%
15	15.0	4.9	67.3%

Graph:



Observations:

Standard Vector Clock Behaviour:

- Shows perfectly linear growth with O(n) complexity
- Average entries per message equals n (as expected)
- Predictable but increasingly expensive as system scales

Singhal-Kshemkalyani Optimization:

- Demonstrates sub-linear growth
- Average entries range from 2.6 to 4.9 (approximately 26-33% of n)
- Maintains strong consistency while significantly reducing overhead

Scalability Analysis:

- The gap between algorithms widens as n increases
- At n=10: SK saves 7.4 entries per message
- At n=15: SK saves 10.1 entries per message
- **Savings increase by ~36% as system scales from 10 to 15 processes**

4. Factors affecting performance

Positive factors (reduce entries):

- Sparse communication patterns
- Lower α values (fewer internal events means less frequent local clock updates)
- Localized communication (processes primarily communicate with neighbours)

Negative factors (increase entries):

- Dense communication graphs
- High α values
- Frequent internal events causing many local updates

For a system with n processes:

- **Best case SK:** O(1) entries when only local clock changes
- **Worst case SK:** O(n) entries when all clocks have been updated
- **Average case SK:** O(k) where k is the average number of processes in causal dependency chains

5. Conclusion

- **Significant Overhead Reduction:** SK optimization reduces message overhead by 67-74% across all tested configurations
- **Scalability:** The optimization becomes increasingly valuable as system size grows