CS5320: Distributed Computing

Programming Assignment 1
Implementing Vector Clocks

Design of the program:

1. **Input Parameters**:

The program reads the following input parameters from a file named "inp-params.txt":

- `n`: Number of nodes (processes) in the system.
- `lambda`: Inter-event time for exponential distribution.
- `alpha`: Ratio of internal to message send events.
- `m`: Total number of messages to be sent by each process.
- Adjacency list representing the communication graph.

2. **Classes and Data Structures**:

`Node` Represents a process in the distributed system.

- Attributes:

- `id`: Unique identifier of the process.
- `vClock`: Vector clock maintained by the process.
- `LS` and `LU`: For SK implementation
- `mLockv`, `mLockLU`, and `mLogFile`: Mutex locks for the things that are used by both the threads
- `sent_messages`: Count of messages sent by the process.
- `done_internal`: Count of internal events executed by the process.
- `neighbors`: List of neighboring processes.
- `markers_received`: Map to track marker messages received from neighbors.
- `vClockEntries`: Size of vector clock entries being sent

- Methods:

- `internal_event()`: Simulates execution of an internal event by the process.
- `send()`: Simulates sending a message to a randomly chosen neighbor.
- `send_marker()`: Sends marker messages to all neighbors.
- `receive()`: Simulates receiving and processing messages.

- Additional global variables and MPI functions are used for communication and synchronization.

4. **Main Function**:

- Reads input parameters and adjacency list from the input file.
- Initializes **MPI** environment and retrieves the rank of the current process.
- Creates a `Node` object representing the current process.
- Spawns sender and receiver threads for each process.
- Joins sender and receiver threads.
- Finalizes MPI environment.

5. **Sending and Receiving Threads**:

Each process have a sender and a receiver thread.

Sender Thread (`SendNInternal`):

- Generates a random sequence of internal and message-send events based on exponential distribution.
- Executes internal events or sends messages accordingly.
- Sleeps for a random duration between events.
- Sends marker message before termination.

Receiver Thread (`Receive`):

- Waits for incoming messages from neighbors.
- Processes regular messages by updating vector clocks.
- Handles marker messages to detect the termination of the distributed computation.

6. **Logging**:

- Logs internal events, message sends, and message receives to an output file named "output.log" with timestamp and vector clock information. Also, logs the total number of vector clock entries sent by each process to an output file called “Vector Clock entries”.

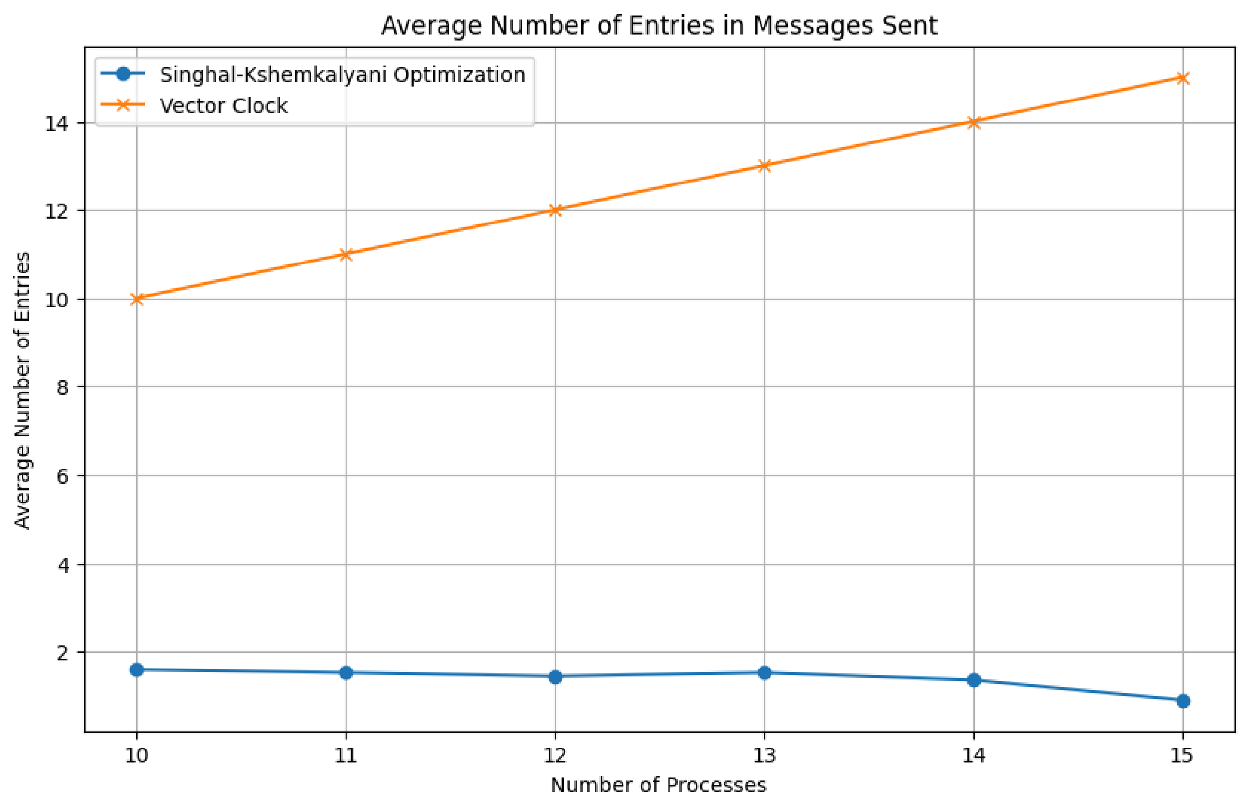
7. **Conclusion**:

- The program terminates after all processes have sent the designated number of messages and received marker messages from all neighbors.
- The output log can be analyzed to study the behavior of the distributed system and evaluate the performance of Vector-clocks and its optimization.

Plot:

Number of processes	The average number of entries in each message sent in the Singhal-Kshemkalyani optimization	The average number of entries in each message sent in Vector Clock
10	1.596	10
11	1.5265	11
12	1.4466	12
13	1.5264	13
14	1.36	14
15	0.910	15

Note: The above values are calculated by taking an average of five runs



The average number of entries sent in each message is the same as the number of processes for Vector clock implementation.

The average number of entries sent in each message remains roughly constant in Singhal-Kshemkalyani Optimization.