Aayush Kumar
CO21BTECH11002
Assignment 1

doWork():

This function simulates the occurrence of events, either internal events or message sends, within each process.
It generates a random number (task) between 0 and 1, then compares it against a threshold (thr) to decide whether to perform an internal event or a message send. This threshold determines the probability of choosing an internal event over a message send event. Since, ratio internal event/message send = thr, and task is a random number between 0 and 1, hence, if we divide the event space from 0 to 1, then internal + send = 1. Hence, the probability of internal event = thr/(1+thr).
Regardless of the type of event (internal event or message send), the function updates the vector clock of the current process (clk[pid-1]++) to reflect the occurrence of the event.
After updating the vector clock, the function logs the details of the event. For internal events, it constructs a log message indicating the event type, process ID, event ID, destination (set to -1 for internal events), timestamp, and vector clock values. For message sends, it constructs a log message containing similar information, including the destination process ID. After logging the event, the function sends the log message to process 0 for centralized logging.
After simulating an event, the function sleeps for a random amount of time.
Once numMessages messages are sent, the function sends a termination message to process 0 and exits.

SK-optimization:

The optimization is implemented by maintaining an additional array called LS (Last Sent) and LU (Last Updated) for each process.
LS[i] keeps track of the last time the vector clock of the current process was sent to the ith process.
LU[i] keeps track of the last time the vector clock entry for process i was updated locally in the current process.

When sending a message to another process, the function checks the LU array to identify which vector clock entries have been updated since the last message sent to the destination process (LS[dest-1]).
Only the updated entries in the vector clock are included in the message payload, reducing the size of the message.
After sending the message, the LS array is updated to reflect the current time (clk[pid-1]), indicating the latest vector clock sent to the destination process.

receiveMsgs():

This function simulates the receipt of messages by each process. It listens for incoming messages from other processes and updates the vector clock and last updated array accordingly.
When a message is received, the function updates the vector clock and last updated array to reflect the receipt of the message. It then constructs a log message containing the event type, process ID, event ID, destination process ID, timestamp, and vector clock values. The log message is sent to process 0 for centralized logging.
The function continues to listen for messages until it receives a termination message from process 0.

main():

The main function initializes the MPI environment, reads input parameters from a file, and assigns roles to processes. The root process (pid = 0) is responsible for logging events and coordinating termination. Other processes simulate events based on their assigned roles and communicate with each other using MPI message passing.
Each process reads its list of edges from the input file and creates two threads to simulate the occurrence of events and the receipt of messages. The first thread calls the doWork function to simulate the occurrence of events, while the second thread calls the receiveMsgs function to simulate the receipt of messages.

Communication Details:

Except for the termination message, all other messages contain some information about the event that occurred and the message that was sent. The structure of the message is as follows:

For send event:
- message[0] - pid of the sending process
- message[1] - message count/id
- message[2] to message[n+1] - vector clock of the sending process (For normal VC)
- message[2] - number of entries being sent (for SK VC)
- message[3] to message[2+message[2]] - vector clock of the sending process (for SK VC)

For log message:
- message[0] - event type
- message[1] - current size of the vector clock
- message[2] - pid of the sending process
- message[3] - message count/id
- message[4] - destination of the message (not applicable for internal events, hence -1)
- message[5] - time of the event
- message[6] to message[n+6] - vector clock of the sending process (For normal VC)
- message[6] - number of entries being sent (for SK VC, not applicable for internal events, receive events, hence 0)
- message[7] to message[n+7] - vector clock of the sending process (For SK VC)

Logging and Termination:

The root process logs the events and sends a termination message to all processes once all messages are sent. The termination message is received by all processes and the receiveMsgs function breaks the loop and terminates the process.

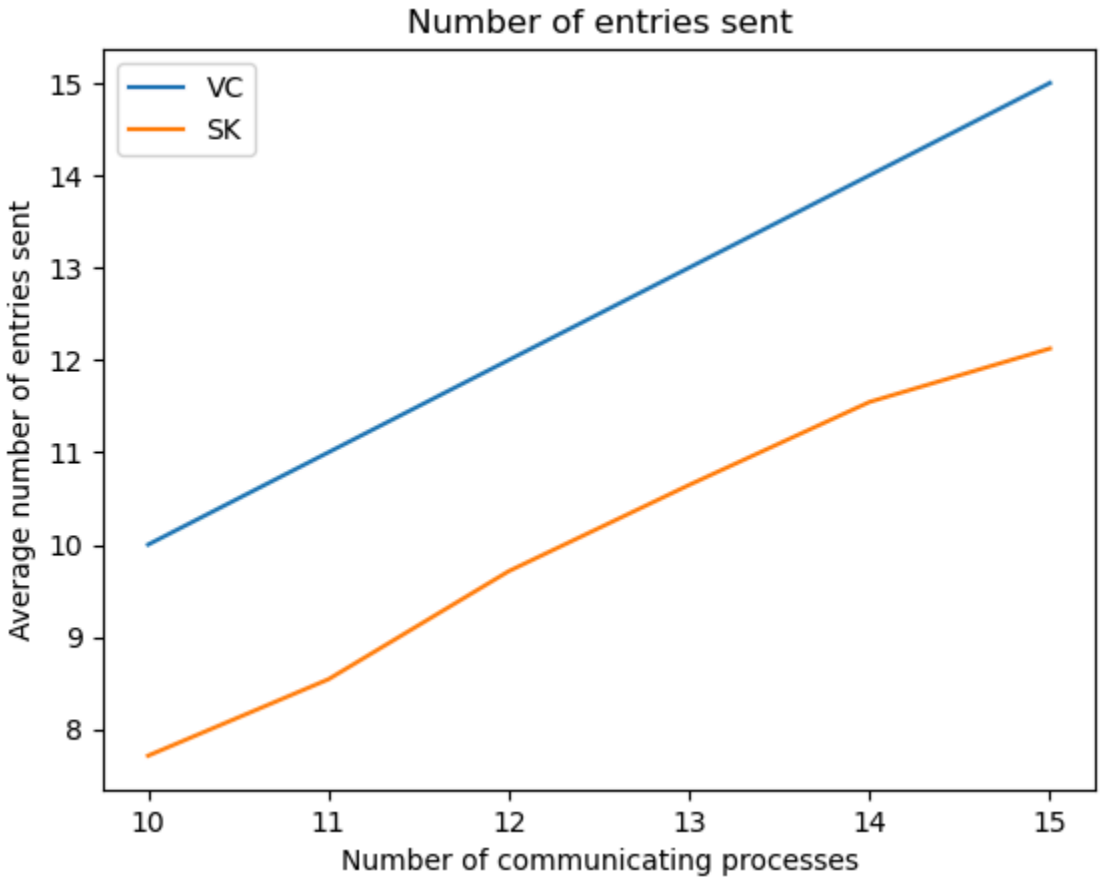
The message/event ids are logged as mi_j/ei_j => j'th message/event from i'th process. The _ is included to avoid confusion in cases like 113 where it can correspond to the 3rd message/event from the 11th process as well as the 13th message/event from the 1st process.

Assumptions:

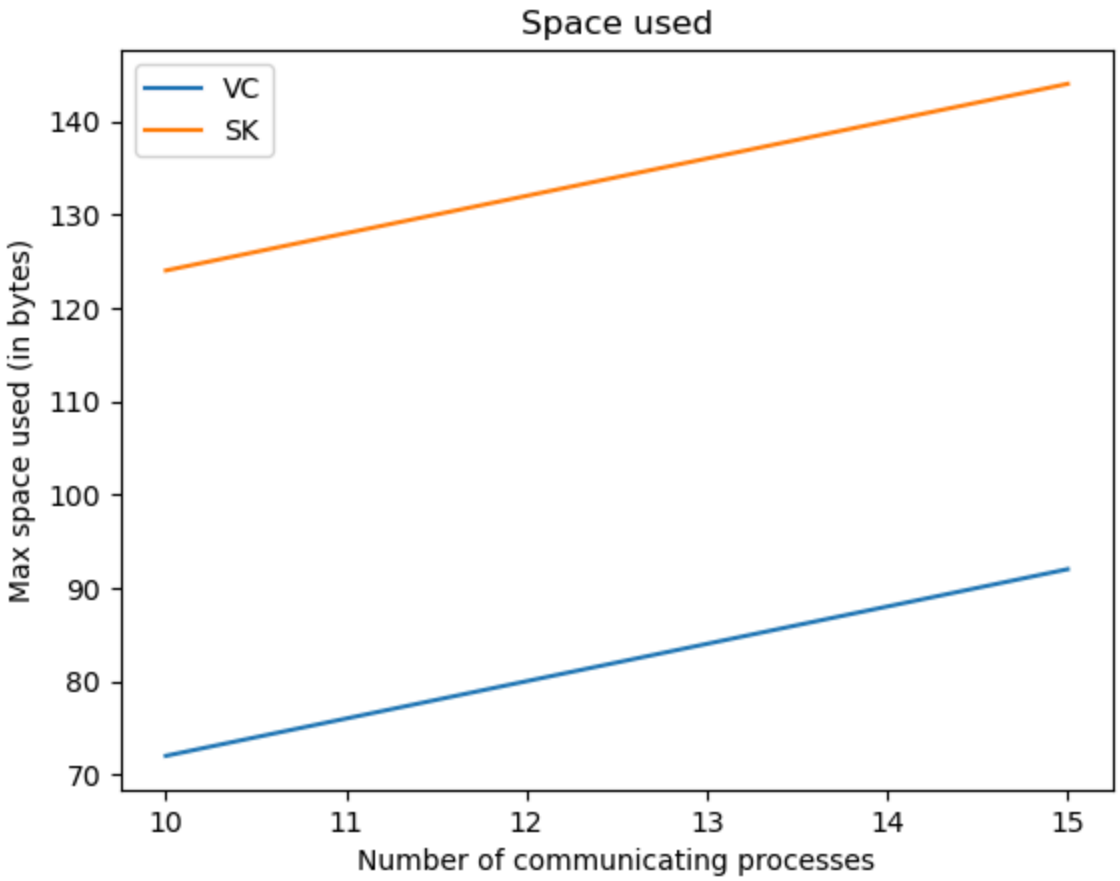
1. The simulation ends when each process sends a fixed number of messages. Hence, it is assumed that each process has at least one outgoing edge to send messages to other processes.

Results:

To keep consistency between experiments, each experiment is performed with a complete graph.



1. As the number of processes increases, the average vector clock entries sent also increases as expected.
2. For SK - optimization, the average vector clock entries sent is less than that for normal vector clock for all values of n.



1. As the number of processes increases, the maximum space used also increases linearly for both implementations.
2. For SK - optimization, the maximum space used is more than that for normal vector clock for all values of n, since we need to store extra information in LS and LU arrays.