DISTRIBUTED SYSTEMS PROJECT

Team No. 14
Shivam Nayak
2018101027

Topic **18**: **Experiment with various deadlock detection algo under various Message delivery guarantees and study the number of messages required to detect the deadlock.**

Language of Implementation: C++ , Python (for plotting comparisons)

Files: (1) MMSR.cpp, (2) CMHAND.cpp, (3) CMHOR.cpp (4)plot.py

Deadlock detection algorithms implemented:

1. **Mitchell and Merritt's Algorithm for the Single-Resource Model.**
2. **Chandy-Misra-Haas's Algorithm for the AND Model.**
3. **Chandy-Misra-Haas's Algorithm for the OR Model.**

Each algorithm is implemented in 3 different message delivery channels.

Message Delivery Channels used in implementation:

1. **Causal Ordering channel**
2. **FIFO channel**

3. Non-FIFO channel

Mitchell and Merritt's Algorithm (Single Resource Model)

Basics of algorithm:

Each process is represented as u/v, where u and v are the public and private labels, respectively. Initially, private and public labels are equal for each process.

A global WFG is maintained and it defines the entire state of the system.The algorithm is defined by the four state transitions. Block creates an edge in the WFG, say between Pa and Pb with public labels u and v respectively, then Pa's labels are replaced with z=inc(u,v), and inc(u,v) yields a unique label greater than both u and v. Two messages are needed: one resource request and one message back to the blocked process to inform it of the public label of the process it is waiting for. Activate denotes that a process has acquired the resource from the process it was waiting for. Transmit propagates larger labels in the opposite direction to the edges by sending a probe message. Whenever a process receives a probe that is less than its public label, it simply ignores that probe. Detect means that the probe with the private label of some process has returned to it, indicating a deadlock. In this way a deadlock is detected.

Implementation of algorithm:

As it's Single resource model, graph is implemented with a simple array (int[ ])

To send transmit messages, transpose of the graph is required, for which vector of arrays is used. (vector<int>[ ])

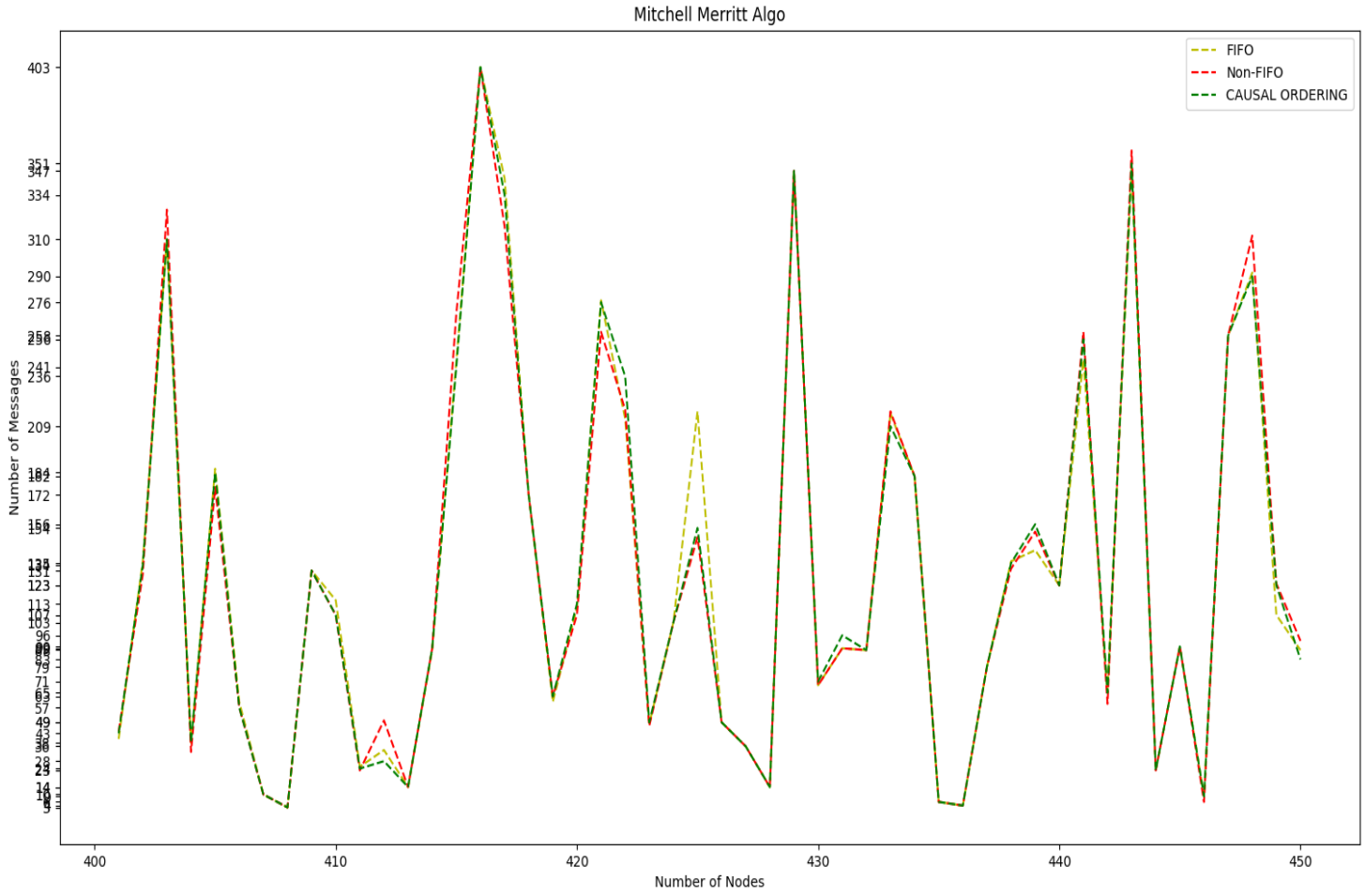
To maintain labels of nodes, another container of pairs is used (vector<pair>)

For each node, it's first ensured if a deadlock will be detected if detection is initiated by it (through DFS). Only if it's confirmed, then all nodes' labels are initialised and the initiating node additionally updates it's label (Block request) and kicks off the chain of transmit messages.
For simulating:

1. Causal Ordering channels → queue data structure is used. (queue<int>)
2. FIFO channels → map data structure is used where the keys are the ids of channels(If edge is from u to v, id is pair(u,v)), and values of the key-value pair are simple queues (queue<int>). (map<pair<int,int>>,queue<int>>)
3. Non-FIFO channels → a simple vector container is used. (vector<int>)

* Labels were initialized before simulating every different channel. Only one execution of Mitchell Merritt's algorithm for every channel takes place for the first node that satisfies the DFS condition, then the loop is discontinued.

Observation:



The algorithm is run from number of nodes 5 to 1000, the whole comparison plot being highly visually vague that's why only from number of nodes 400 to 450 is shown above.

As can be seen the number of messages is not exactly the same for all three message delivery channels.

Chandy-Misra-Haas's Algorithm (AND Model)

Basics of algorithm:

The algorithm uses a special message called probe, which is a triplet (i,j,k), denoting that it belongs to a deadlock detection initiated for process Pi and it is being sent by the home site of process Pj to the home site of process Pk. A probe message travels along the edges of the global WFG graph, and a dead-lock is detected when a probe message returns to the process that initiated it.

Implementation of algorithm:

As it's for the AND model and thus outdegree of nodes can be more than 1, graph is implemented with a vector container of sets. (vector<set<int>>)

For each node, it's first ensured if a deadlock will be detected if detection is initiated by it (through DFS), as no termination detection is implemented and if we initiate a deadlock from a node which is not deadlocked, the algorithm will continue infinitely. Also as graphs are generated pseudo-randomly, it's not practical to ensure the existence of deadlock from a node beforehand during generation of graphs. Only if it's confirmed that the node is stuck in a deadlock, we kick off the algorithm.

Even though in theory, the probe is a triplet, in this implementation I have only sent a pair as a probe because our only concern is the detection of deadlock and the number of probes required for it, and for this purpose a pair probe is sufficient.

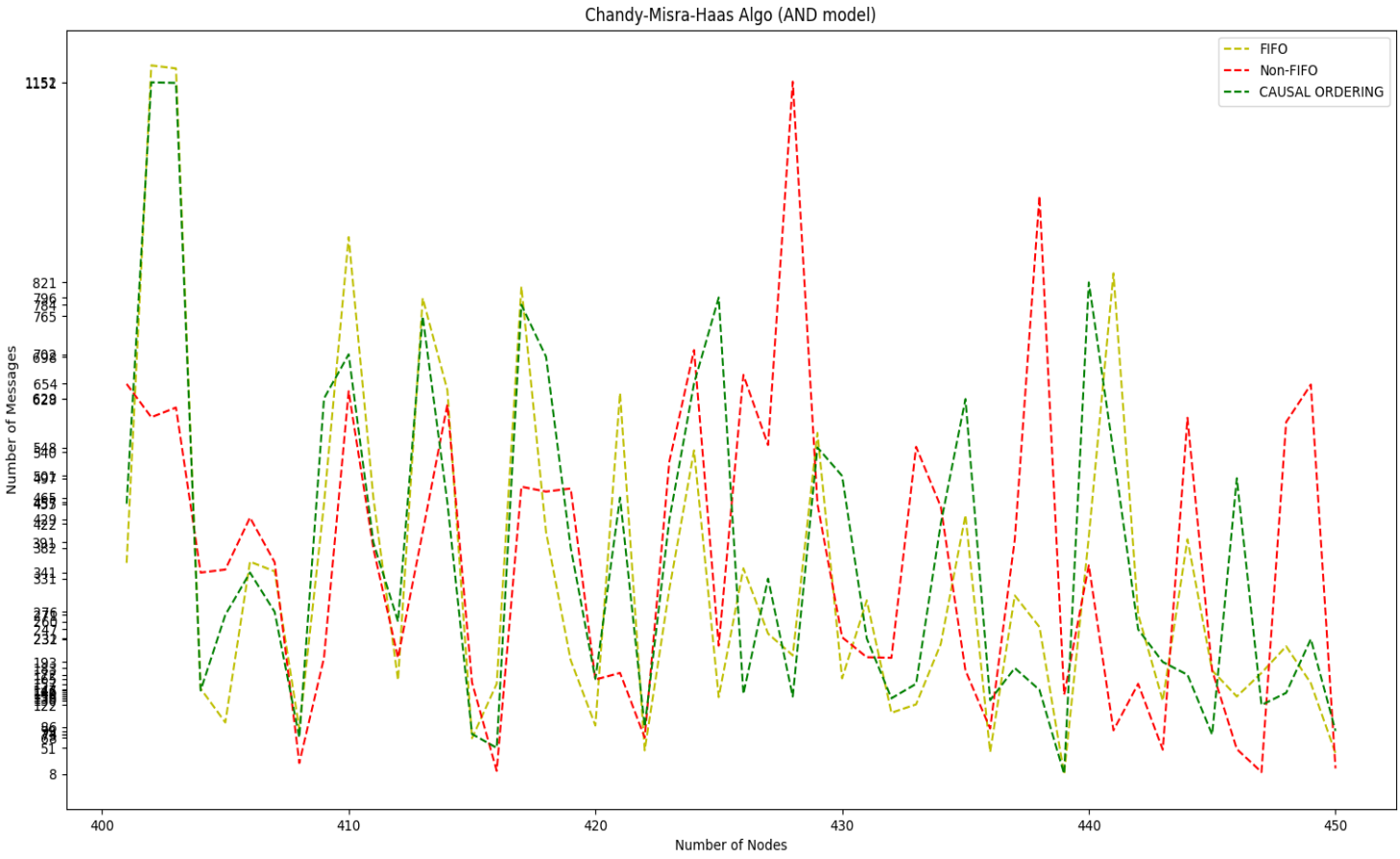
For simulating:

4. Causal Ordering channels → queue data structure is used. (queue<pair<int,int>>)

5. FIFO channels → map data structure is used where the keys are the ids of channels(If edge is from u to v, id is pair(u,v)), and values of the key-value pair are simple queues (queue<pair<int,int>>).
   (map<pair<int,int>>,queue<pair<int,int>>>)
6. Non-FIFO channels → a simple vector container is used. (vector<pair<int,int>>)

* Only one execution of the algorithm for every channel takes place for the first node that satisfies the DFS condition, then the loop is discontinued.

Observation:



The algorithm is run from number of nodes 5 to 1000, the whole comparison plot being highly visually vague that's why only from number of nodes 400 to 450 is shown above.

The difference in the number of probes for the different channels is significantly more than Mitchell and Merritt's Algorithm.

Chandy-Misra-Haas's Algorithm (OR Model)

Basics of algorithm:

A blocked process initiates deadlock detection by sending query messages to all processes in its dependent set (i.e., processes from which it is waiting to receive a message). If an active process receives a query or reply message, it discards it. When a blocked process Pk receives a query(i,j,k) message, it takes the following actions:

1. If this is the first query message received by Pk for the deadlock detection initiated by Pi, then it propagates the query to all the processes in its dependent set.
2. Else, Pk returns a reply message to it immediately provided Pk has been continuously blocked since it received the corresponding engaging query. Otherwise, it discards the query.

The initiator process detects a deadlock when it has received reply messages to all the query messages it has sent out.

Implementation of algorithm:

As it's for the OR model and thus outdegree of nodes can be more than 1, graph is implemented with a vector container of sets. (vector<set<int>>)

For each node, it's first ensured if a deadlock will be detected if detection is initiated by it (through a KNOT detection algorithm), as no termination detection is implemented and if we initiate a deadlock from a node which is not deadlocked, the algorithm will continue

infinitely. Also as graphs are generated pseudo-randomly, it's not practical to ensure the existence of deadlock from a node beforehand during generation of graphs. Only if it's confirmed that the node is stuck in a deadlock, we kick off the algorithm.

Even though in theory, the messages (query and reply) in this algorithm are triplet, in this implementation I have only sent a pair as a message because our only concern is the detection of deadlock and the number of messages required for it, and for this purpose a pair message is sufficient.

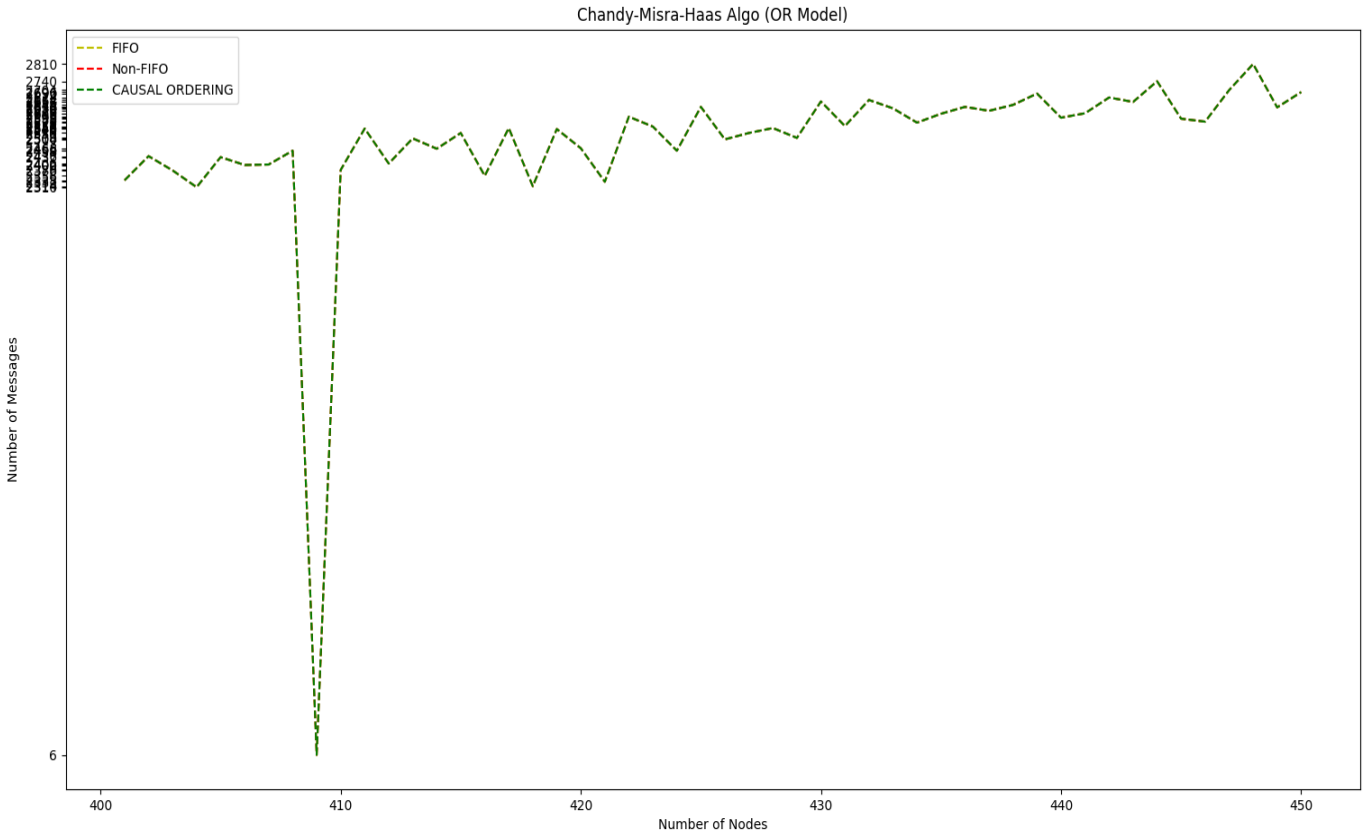
Also a point to note here is if there is a deadlock, the number of query messages will be equal to the number of reply messages. As our concern is just the number of messages required to detect the deadlock, (and we have already ensured there is a deadlock including the initiator node) sending replies would be irrelevant in regard to our objective. That's why once the number of query messages are determined, the implemented version of the algorithm comes to a stop.

For simulating:

7. Causal Ordering channels → queue data structure is used. (queue<pair<int,int>>)
8. FIFO channels → map data structure is used where the keys are the ids of channels(If edge is from u to v, id is pair(u,v)), and values of the key-value pair are simple queues (queue<pair<int,int>>).
   (map<pair<int,int>>,queue<pair<int,int>>>)
9. Non-FIFO channels → a simple vector container is used. (vector<pair<int,int>>)

* Only one execution of the algorithm for every channel takes place for the first node that satisfies the Knot detection condition, then the loop is discontinued.

Observation:



The algorithm is run from number of nodes 5 to 1000, the whole comparison plot being highly visually vague that's why only from number of nodes 400 to 450 is shown above.

As can be seen from the plot, the number of probes for detecting deadlock is the same for every channel unlike previous algorithms. This is not specific for this range only, it is a property of this algorithm which is used for the OR model.

Generating Pseudo-Random Graphs

The skeleton of the random graph was built by connecting every node to a random node with indegree 0, ensuring outdegree 1 for each node. Thus ensuring a cycle in the graph and Single resource model.
So this graph is used for Mitchell Merritt's algorithm for Single resource (as a cycle in Single resource model graph ensures deadlock).

Building upon this skeleton (in which every node has outdegree 1), additional nodes are connected to each node (number of additional nodes is randomized). The connected additional nodes are also randomized.

As there is already a cycle in this graph, it would surely have a deadlock in the AND model, so we can use this graph for Chandy Misra Haas algorithm for AND model.

After executing the code for many times, it was easy to see that knots are also created within this graph almost for all varying numbers of nodes. Hence this graph would also have a deadlock in the OR model, thus can also be used for the Chandy Misra Haas algorithm for the OR model.

LINK TO THE VIDEO:
https://iiitaphyd-my.sharepoint.com/:v:/g/personal/shivam_nayak_students_iiit_ac_in/EQWg1RBowyFBgWQvSbOTBd4BM3bY56muxbxvRcyff2FBJw