Spanning Tree based Termination Detection

Aditya Saripalli (20173071)

Issac Balaji (20163051)

Pre-Work

01. Simple Algorithm

A token-based algorithm when all children are done terminated parent is terminated.

02. Rodney Topor's

Color based token which address the issue in simple algo.

03. Chandrasekaran And Venkatesan’s

Distributed Termination Algorithm using message optimal termination detection

04. Arora Gupta's

Distributed Termination Algorithm by 2 phases detection and temination

05. Our Work

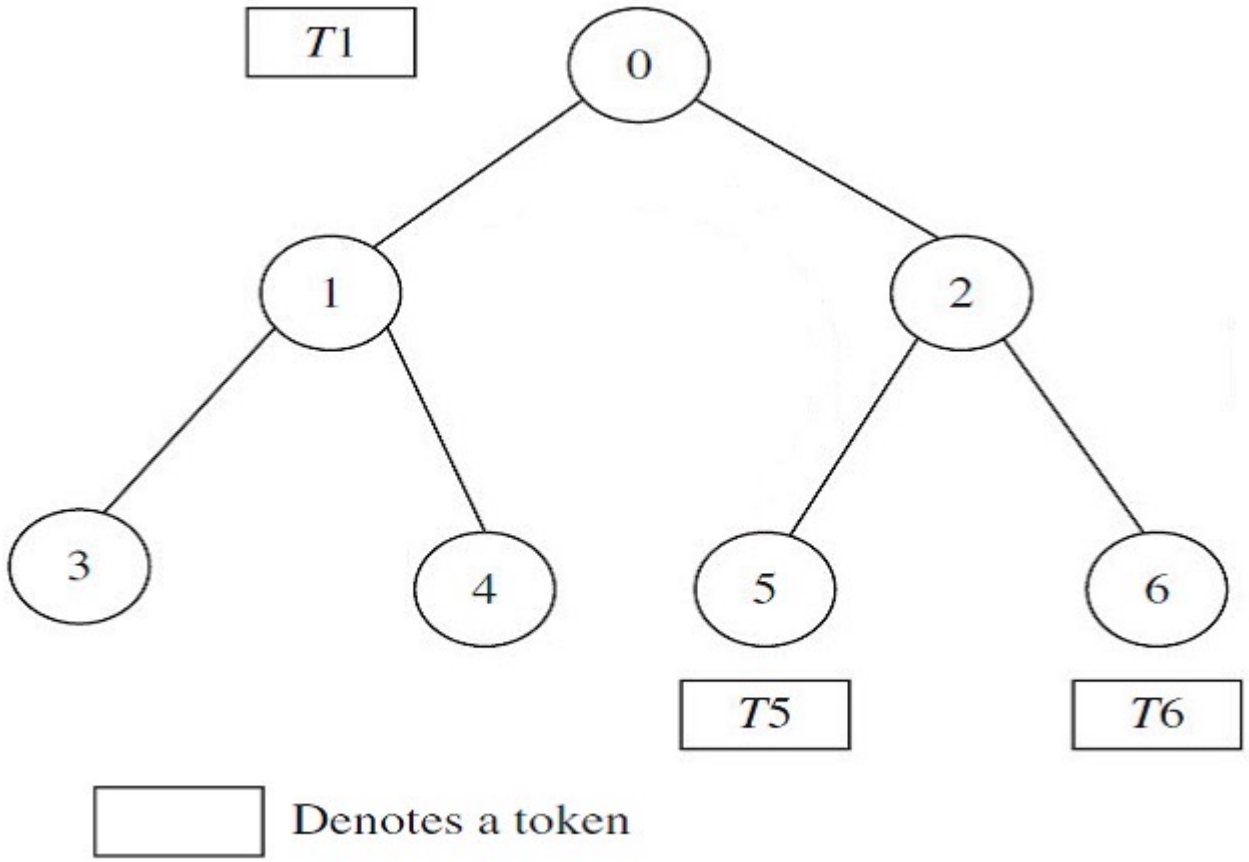
Used Rodney Topor's algo with our change in algo.

Assumptions

- Minimum spanning tree (path) is known
- Nodes are available and not modified.
- No new channels (other than the edges of the MST) are established.

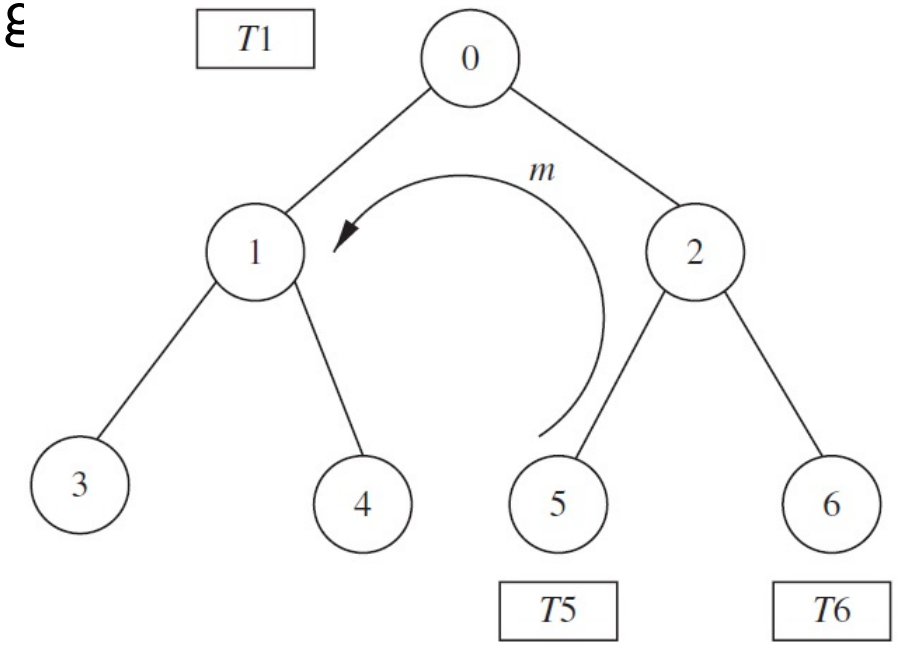
Simple Algorithm

- N processes P_i , $0 \leq i \leq N$, which are modeled as the nodes i
- edges of the graph represent the communication channels.
- Children report to their parents, if they have terminated.
- parent node will similarly report to its parent when it has completed processing and all of its immediate children have terminated
- Algorithm terminated when root terminates.



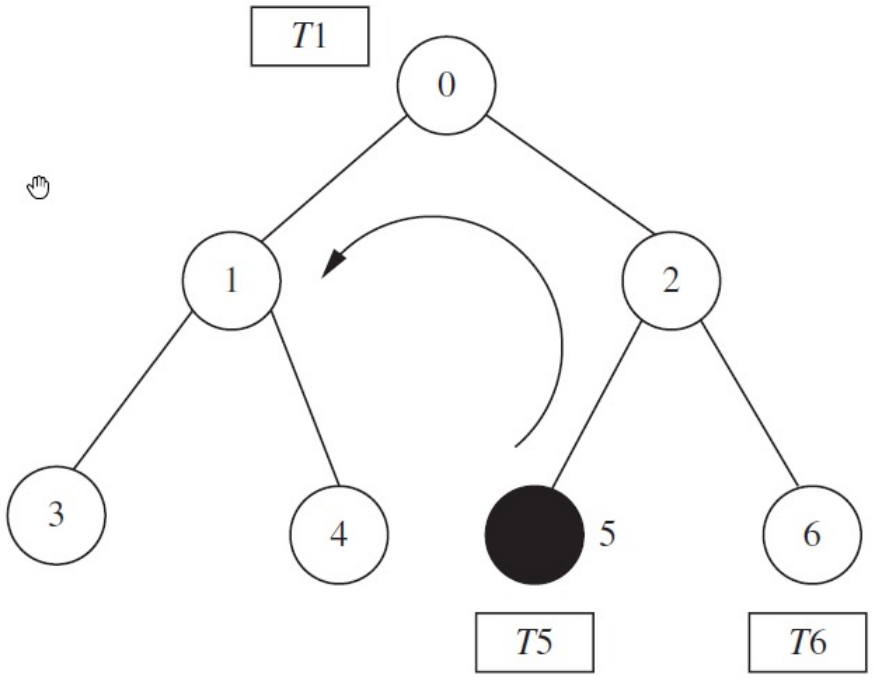
Problem with the algorithm:

The algorithm fails when a process (after it has sent a token to its parent), receives a messag from some other process.



Rodney.W.Topor's

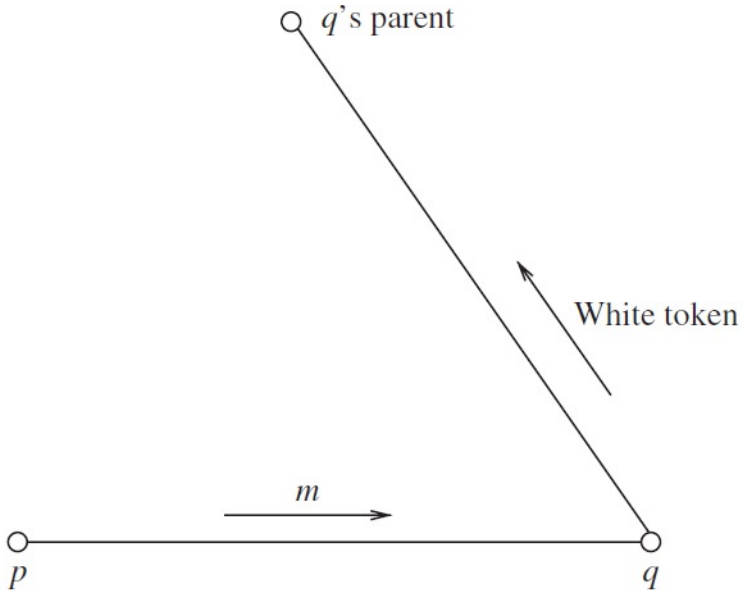
- Initially color all the processes and tokens as WHITE.
- A process turns BLACK when it sends a message to some other process. It turns WHITE, after it has sent the BLACK token to its parent.
- Upon receiving a BLACK token (from one of the child(s)) Root will send a REPEAT signal to all its children propagating till leaf node.
- The leaf nodes then restart the algorithm on receiving the REPEAT signal.
- Root node concludes that termination detection is complete only on receiving WHITE tokens from all the child nodes.



Performance

- Best case Message Complexity O(N)
  - One Round
- Worst case Message Complexity O(N*M)
  - M – no of computation messages exchanged on black token
- Best case - when the token needs to be sent to its parent so N nodes will lead to O(N) complexity.
- Worst case - if it takes M no of rounds of tokens passing. Then M times N nodes must communicate and marks for complexity O(N * M).

S.Chandrasekaran And S.Venkatesan's



- An extension of Rodney.W.Topor.
- When a node *p* sends a message *m* to node *q*, *p* should wait until ***q*** becomes idle.
- When the node *q* terminates, it sends an acknowledgement (a CONTROL message) to node *p* informing node *p*
- Both the sender and the receiver keep track of each message exchange.
- All nodes will only send WHITE token.
- A message optimal way of termination detection.

Performance

Topor's model

- Worse Case Message Complexity O(N*M)

Message Optimal

- the total number of messages generated by the algorithm is 2* |E| + |V| − 1 + M.
  - E edges – links / warning messages
  - M remove message
  - V nodes
- Message Complexity O(|E| + M)
  - as |E| > |V|

R.K.Arora and M.N.Gupta's

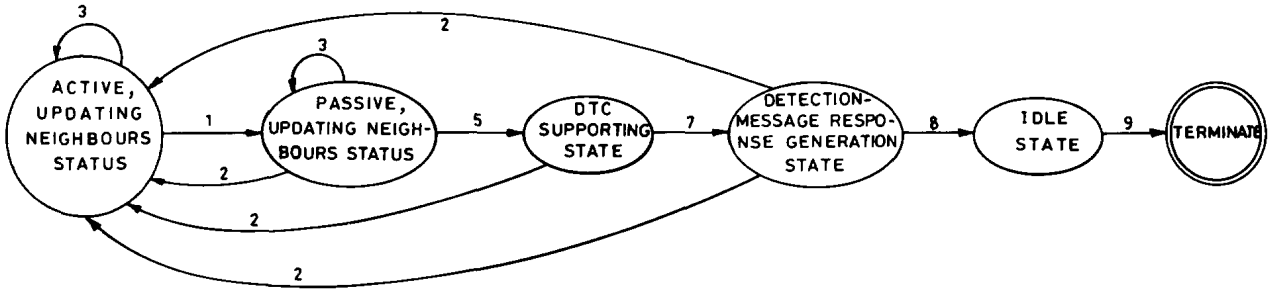


Fig. 1. State transition diagram of a leaf process p_i ($i \neq 1$).

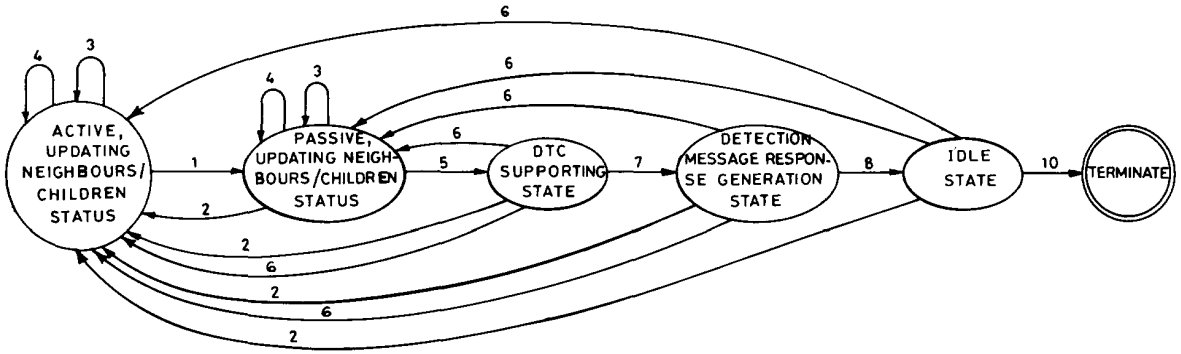


Fig. 2. State transition diagram of an internal process p_i ($i \neq 1$).

R.K.Arora and M.N.Gupta's (contd...)

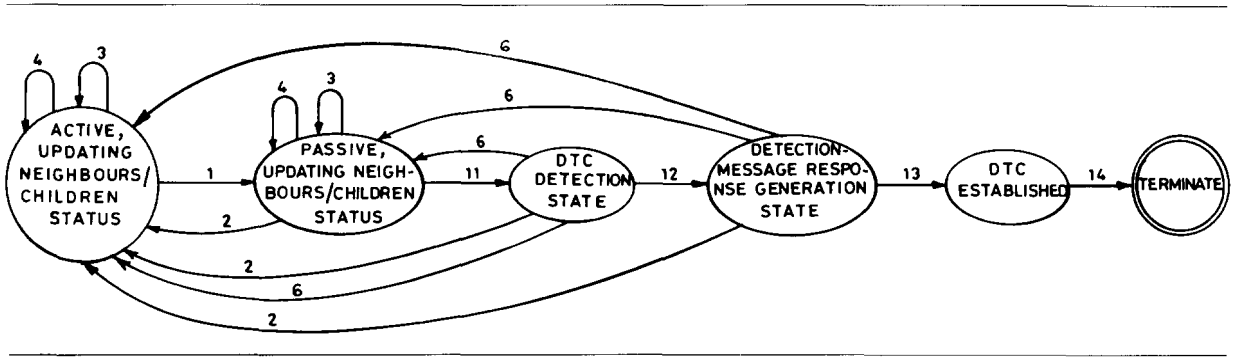


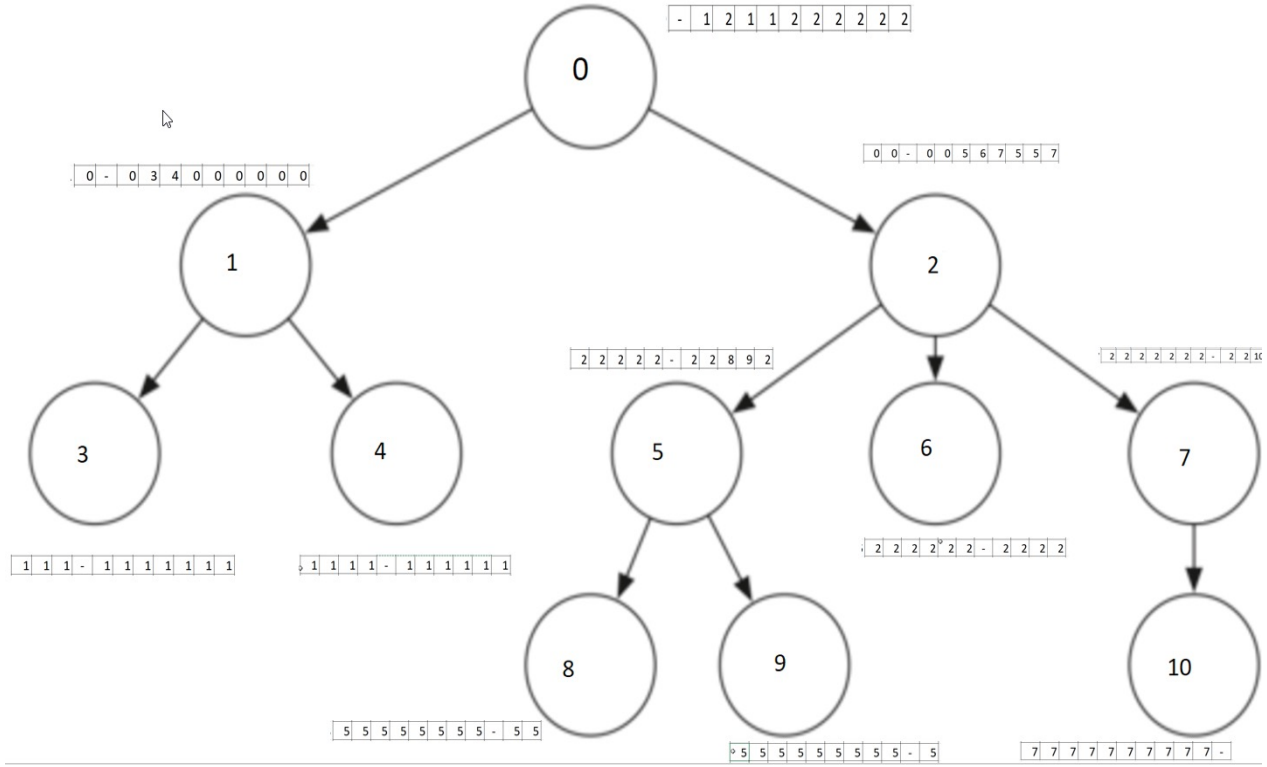
Fig. 3. State transition diagram of the root process p_i

R.K.Arora and M.N.Gupta's (contd…)

- The detection-message is issued only when the process and all its communicating neighbors become PASSIVE.
- The detection message wave once issued by the root process first spreads downwards and then contracts upwards.
- No additional effort is required to maintain the local information in the control section of a process.

Our Model

- We have used Rodney.W.Topor's model for termination detection.
- In addition to it, we have added an algorithm for computing a routing array for message passing.
- Messages are sent only along the edges of the nodes using the routing path mentioned in the array.



Comparision

Rodney.W.Topor

- Simple
- More messages
- Frequent Repeats
- Complete env repeats for even one black token

Message Optimal

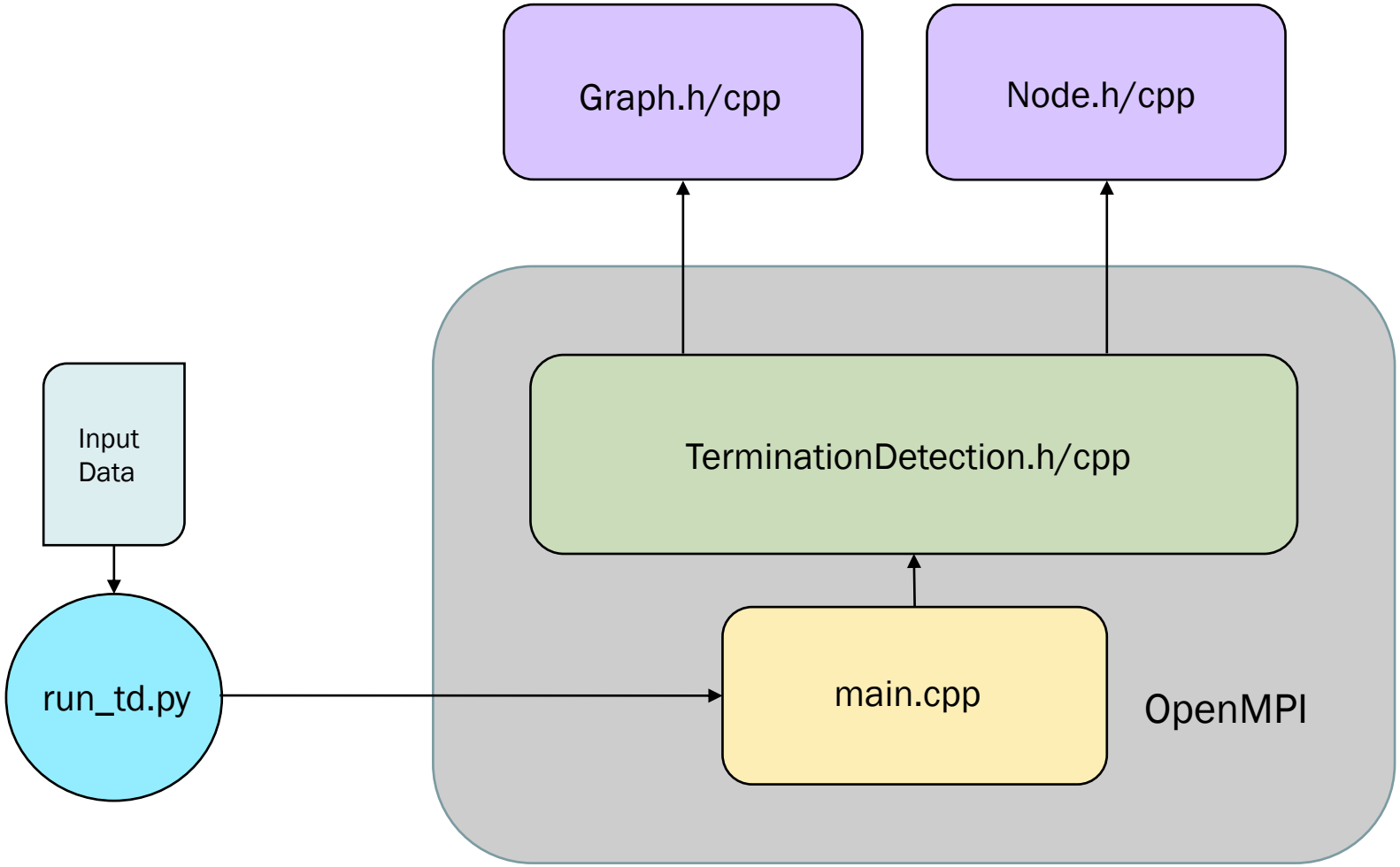
- Less message overheads and message traffic
- Mostly waiting for other nodes to go to idle.
- In Arora’s method even waiting for neighborhood nodes to become passive

Our model

- Comparatively Simple after initialization.

Our Model - A Deep Dive
(Design & Implementation)

Design


Graph.h/cpp
Node.h/cpp
Input Data
TerminationDetection.h/cpp
run_td.py
main.cpp
OpenMPI

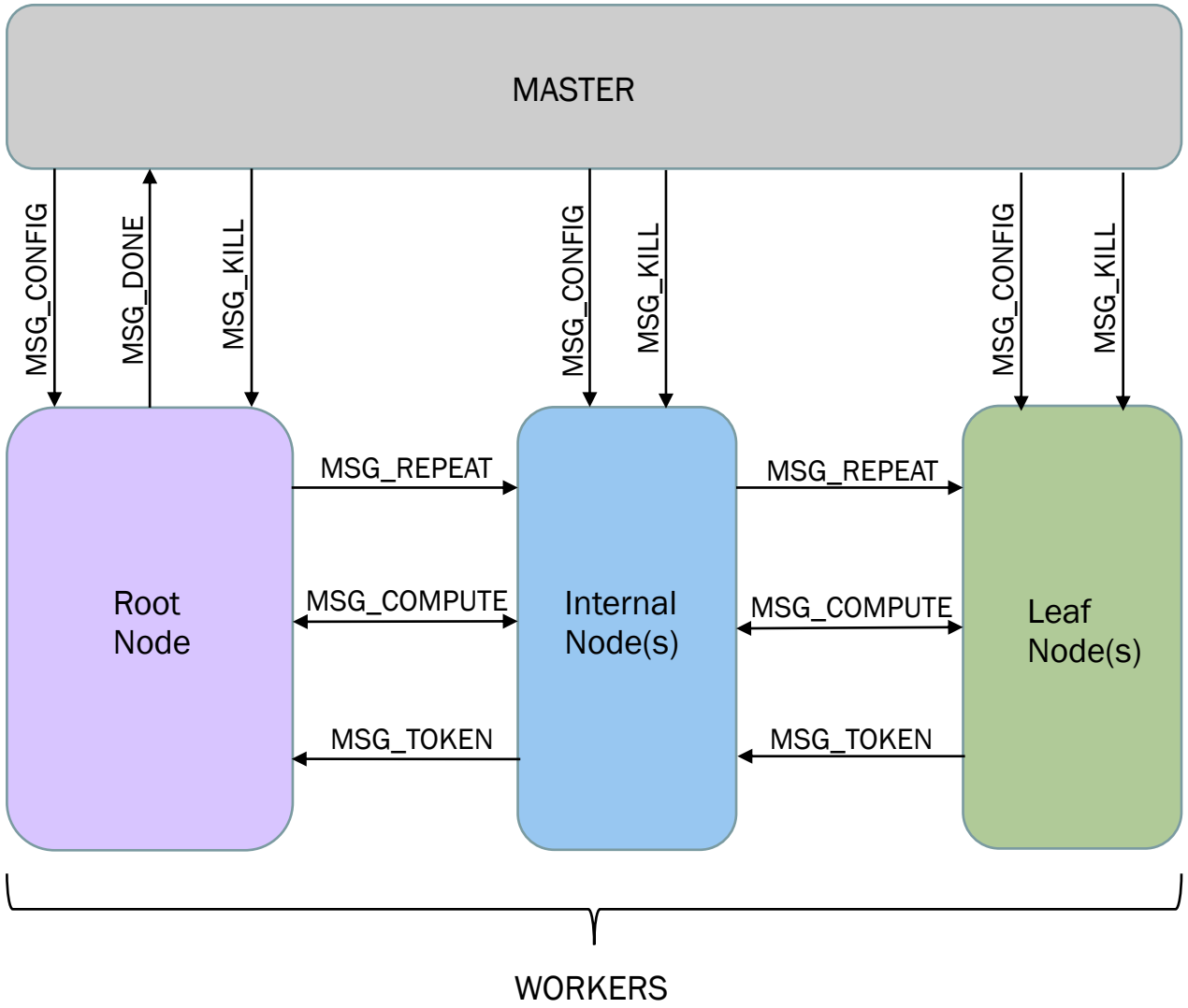
Implementation

- For “N” nodes we create (N+1) OpenMPI processes.
- Process with Rank=0 will be a master/manager process.
- Processes with ranks 1 to N will represent N nodes of the MST.
- As naming convention, we call master/manager process as MASTER and others as WORKER processes respectively.
- In OpenMPI, we only use MPI_COMM_WORLD communicator for blocking send and receive communications between processes.
- Messages supported : MSG_CONFIG, MSG_DONE, MSG_KILL, MSG_COMPUTE, MSG_REPEAT, MSG_TOKEN.
- Types of nodes : RootNode, InternalNode, LeafNode

Implementation (contd..)

Type of Process	Type of Node (in the MST)	Messages involved in ...	
		Sending	*Receiving*
MASTER	--	MSG_CONFIG, MSG_KILL	MSG_DONE
WORKER	RootNode	MSG_REPEAT, MSG_COMPUTE, MSG_DONE	MSG_CONFIG, MSG_KILL, MSG_COMPUTE, MSG_TOKEN
	InternalNode	MSG_REPEAT, MSG_COMPUTE, MSG_TOKEN	MSG_CONFIG, MSG_KILL, MSG_REPEAT, MSG_COMPUTE, MSG_TOKEN
	LeafNode	MSG_TOKEN, MSG_COMPUTE	MSG_CONFIG, MSG_KILL, MSG_REPEAT, MSG_COMPUTE

Message passing between nodes/processes



MASTER

- Reads the input data file containing the graph.
- Instantiate a Graph object and saved the input graph in it, for further computations.
- Runs Kruskal's algorithm with a Union-Find data structure to compute the MST as an adjacency list.
- From the adjacency list identify the list of child nodes for each node in the spanning tree.
- Computes the routing table for the MST for nodes to send messages among themselves.
- For each node in the MST send the RootNode, ChildNodes and Routing Array, specific to that node/process only.
- Generate a random compute message, with a source & destination selected randomly, and send it to the source node WORKER process.
- Wait for MSG_DONE message from the RootNode indicating the termination detection algorithm is completed.
- Send MSG_KILL message to all the WORKER processes to terminate them gracefully.

WORKER

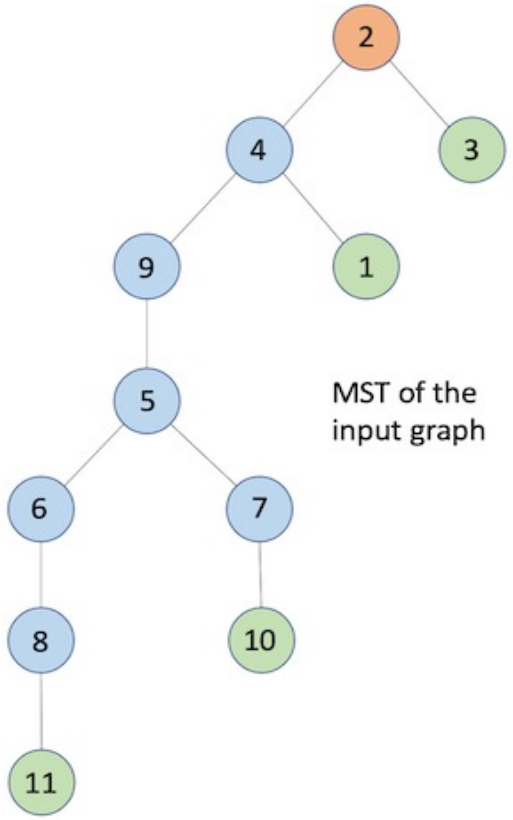
- Instantiate a Node object.
- Receive all the MSG_CONFIG messages from the MASTER & store them in the Node object.
- If the current WORKER process rank matches the COMPUTE message source node, then the compute message is saved, otherwise discarded.
- The WORKER node then start executing a computations loop (which would randomly take any time between 1 to 5 seconds).
- During the computations, the node will check if it has any saved compute message.
- If its there, then it sends the messages using the routing array, and mark its token color as BLACK. Now it is a BLACK process.
- All the leaf WORKER nodes, after their respective computations are done, will start the termination detection by sending MSG_TOKEN to their parent node.
- All the internal nodes will wait until they have received MSG_TOKEN messages from all their child nodes.

WORKER (contd..)

- Once received if there is a BLACK token among them, then forward the same to the parent node. Otherwise, send a WHITE token to the parent node.
- A BLACK process, after sending it BLACK token to the parent node, will mark its token as WHITE.
- The Root WORKER node will wait for the tokens from all the child nodes.
- Once received if there is a BLACK token in them, then it will initiate a REPEAT signal to all the child nodes.
- Once this REPEAT signal reaches the leaf node, the leaf node will re-initiate termination detection algorithm again.
- After the Root node has received all the WHITE tokens from all its child nodes, it will send MSG_DONE message to the MASTER process.
- Once the message MSG_KILL is received – stop and exit the WORKER process.

Sample Input Data

```
#####################################################
# Input file format for the creation of graph
# <no_of_nodes>
# <source_vertex> <destination_vertex> <edge_weight>
# <source_vertex> <destination_vertex> <edge_weight>
# ...
#####################################################
11
1 2 10
1 3 7
1 4 6
2 3 4
2 4 3
2 5 8
2 6 11
3 8 6
4 9 5
5 6 5
5 7 5
5 9 2
6 8 2
6 11 13
7 10 6
7 11 9
8 11 7
9 10 17
```



MST of the input graph

Sample Routing Table

	1	2	3	4	5	6	7	8	9	10	11
1	-1	4	4	4	4	4	4	4	4	4	4
2	4	-1	3	4	4	4	4	4	4	4	4
3	2	2	-1	2	2	2	2	2	2	2	2
4	1	2	2	-1	9	9	9	9	9	9	9
5	9	9	9	9	-1	6	7	6	9	7	6
6	5	5	5	5	5	-1	5	8	5	5	8
7	5	5	5	5	5	5	-1	5	5	10	5
8	6	6	6	6	6	6	6	-1	6	6	11
9	4	4	4	4	5	5	5	5	-1	5	5
10	7	7	7	7	7	7	7	7	7	-1	7
11	8	8	8	8	8	8	8	8	8	8	-1

Sample path from : Node[11] --------> Node[2]

Execution Run

```
implementation$ ./run_td.py test/input_1.txt
[INFO] Building binaries ...
[MAKE] Cleaning all the object files and binaries.
[MAKE] Compiled src/Graph.cpp successfully.
[MAKE] Compiled src/Node.cpp successfully.
[MAKE] Compiled src/TerminationDetection.cpp successfully.
[MAKE] Compiled src/main.cpp successfully.
[MAKE] Linking Complete.
[INFO] No of Nodes in the given graph: 11
[INFO] Initiating Termination Detection with 12 processes (1 process per node and 1 additional master/manager process)
[INFO] MASTER Process configuring and setting the process(s) environment
[INFO] Displaying MST of the given graph as an Adjacency List:
1 -> 4
2 -> 4,3
3 -> 2
4 -> 2,9,1
5 -> 9,6,7
6 -> 8,5
7 -> 5,10
8 -> 6,11
9 -> 5,4
10 -> 7
11 -> 8
[INFO] Root Node: 2
```

Execution Run (contd..)

```
[INFO] Root Node: 2
[INFO] Node[4] is done with internal computations
[INFO] Node[8] is done with internal computations
[INFO] Node[5] is done with internal computations
[INFO] Node[7] is done with internal computations
[INFO] Node[6] is done with internal computations
[INFO] Node[3] is done with internal computations
[INFO] LeafNode[3] initiating Termination Detection
[INFO] Node[9] is done with internal computations
[INFO] Node[2] is done with internal computations
[INFO] RootNode[2] Received Token[1] from ChildNode[3]
[INFO] Node[10] is done with internal computations
[INFO] LeafNode[10] initiating Termination Detection
[INFO] InternalNode[7] Received Token[1] from ChildNode[10]
[INFO] InternalNode[7] Received all tokens from child nodes. Sending Token[1] to ParentNode[5]
[INFO] InternalNode[5] Received Token[1] from ChildNode[7]
[INFO] Node[11] is done with internal computations
[INFO] LeafNode[11] initiating Termination Detection
[INFO] InternalNode[8] Received Token[1] from ChildNode[11]
[INFO] InternalNode[8] Received all tokens from child nodes. Sending Token[1] to ParentNode[6]
[INFO] InternalNode[6] Received Token[1] from ChildNode[8]
[INFO] InternalNode[6] Received all tokens from child nodes. Sending Token[1] to ParentNode[5]
[INFO] Node[1] sent a COMPUTE message to Node[6]
[INFO] Node[1] is done with internal computations
[INFO] LeafNode[1] initiating Termination Detection
[INFO] InternalNode[5] Received Token[1] from ChildNode[6]
[INFO] InternalNode[5] Received all tokens from child nodes. Sending Token[1] to ParentNode[9]
[INFO] InternalNode[6] Received the COMPUTE message from Node[1]
[INFO] InternalNode[4] Received Token[0] from ChildNode[1]
[INFO] InternalNode[9] Received Token[1] from ChildNode[5]
[INFO] InternalNode[9] Received all tokens from child nodes. Sending Token[1] to ParentNode[4]
[INFO] InternalNode[4] Received Token[1] from ChildNode[9]
[INFO] InternalNode[4] Received all tokens from child nodes. Sending Token[0] to ParentNode[2]
```

COMPUTE message sent

COMPUTE message received

Node 1 sent BLACK token to Node 4

Execution Run (contd..)

Root node received BLACK token. Sending REPEAT to all child nodes

Leaf nodes re-initiating termination detection.

Node 1 sent WHITE token to Node 4

```
[INFO] InternalNode[4] Received all tokens from child nodes. Sending Token[0] to ParentNode[2]
[INFO] RootNode[2] Received Token[0] from ChildNode[4]
[INFO] RootNode[2] Received a BLACK token. Initiating REPEAT Signal
[INFO] RootNode[2] Sent REPEAT Signal to ChildNode[4]
[INFO] RootNode[2] Sent REPEAT Signal to ChildNode[3]
[INFO] LeafNode[1] Received a REPEAT request from ParentNode[4]
[INFO] LeafNode[1] initiating Termination Detection
[INFO] RootNode[2] Received Token[1] from ChildNode[3]
[INFO] LeafNode[3] Received a REPEAT request from ParentNode[2]
[INFO] LeafNode[3] initiating Termination Detection
[INFO] InternalNode[4] Received a REPEAT request from ParentNode[2]
[INFO] InternalNode[4] Forwarding REPEAT signal to ChildNode[9]
[INFO] InternalNode[4] Forwarding REPEAT signal to ChildNode[1]
[INFO] InternalNode[4] Received Token[1] from ChildNode[1]
[INFO] InternalNode[9] Received a REPEAT request from ParentNode[4]
[INFO] InternalNode[9] Forwarding REPEAT signal to ChildNode[5]
[INFO] InternalNode[5] Received a REPEAT request from ParentNode[9]
[INFO] InternalNode[5] Forwarding REPEAT signal to ChildNode[6]
[INFO] InternalNode[5] Forwarding REPEAT signal to ChildNode[7]
[INFO] InternalNode[6] Received a REPEAT request from ParentNode[5]
[INFO] InternalNode[6] Forwarding REPEAT signal to ChildNode[8]
[INFO] InternalNode[7] Received a REPEAT request from ParentNode[5]
[INFO] InternalNode[7] Forwarding REPEAT signal to ChildNode[10]
[INFO] InternalNode[7] Received Token[1] from ChildNode[10]
[INFO] InternalNode[7] Received all tokens from child nodes. Sending Token[1] to ParentNode[5]
[INFO] InternalNode[8] Received a REPEAT request from ParentNode[6]
[INFO] InternalNode[8] Forwarding REPEAT signal to ChildNode[11]
[INFO] InternalNode[5] Received Token[1] from ChildNode[7]
[INFO] InternalNode[8] Received Token[1] from ChildNode[11]
[INFO] InternalNode[8] Received all tokens from child nodes. Sending Token[1] to ParentNode[6]
[INFO] LeafNode[10] Received a REPEAT request from ParentNode[7]
[INFO] LeafNode[10] initiating Termination Detection
[INFO] LeafNode[11] Received a REPEAT request from ParentNode[8]
[INFO] LeafNode[11] initiating Termination Detection
[INFO] InternalNode[5] Received Token[1] from ChildNode[6]
```

Execution Run (contd..)

```
[INFO] InternalNode[5] Received Token[1] from ChildNode[6]
[INFO] InternalNode[5] Received all tokens from child nodes. Sending Token[1] to ParentNode[9]
[INFO] InternalNode[6] Received Token[1] from ChildNode[8]
[INFO] InternalNode[6] Received all tokens from child nodes. Sending Token[1] to ParentNode[5]
[INFO] InternalNode[9] Received Token[1] from ChildNode[5]
[INFO] InternalNode[9] Received all tokens from child nodes. Sending Token[1] to ParentNode[4]
[INFO] RootNode[2] Received Token[1] from ChildNode[4]
[INFO] RootNode[2] Received all tokens from child nodes
[INFO] InternalNode[4] Received Token[1] from ChildNode[9]
[INFO] InternalNode[4] Received all tokens from child nodes. Sending Token[1] to ParentNode[2]
[INFO] Termination Detection completed
```

References & Project Links

- Termination Detection for Distributed Computations – *Rodney.W.Topor*
- A Message-Optimal Algorithm for Distributed Termination Detection – *S.Chandrashekharan & S.Venkatesan*
- An Algorithm for Solving Distributed Termination Detection – *R.K.Arora & M.N.Gupta*
- Distributed Computing – Principles, Algorithms & Systems (Chapter 7) – *Ajay D. Kshemkalyani & Mukesh Singhal*
- MPI The Complete Reference - *Marc Snir, Steve Otto, Steven Huss Lederman, David Walker, Jack Dongarra*
- OpenMPI 4.1.1 Documentation - https://www.open-mpi.org/doc/current/
- Project GitHub Page (will be made public after 30th April, 11:55 PM) https://github.com/adisarip/DS_Project
- Presentation Video https://github.com/adisarip/DS_Project/blob/main/presentation/final_presentation_video.mp4

Thank you