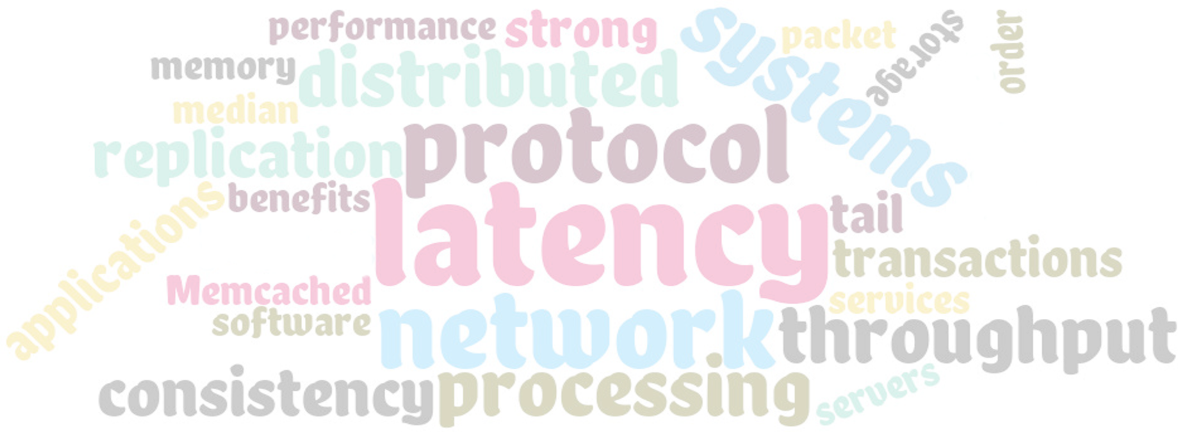


Self-Stabilization in Distributed Systems

Course: Distributed Computing

Faculty: Dr. Rajendra Prasath

Spring 2019

About this topic

This course covers various concepts in **Self-Stabilization in Distributed Systems.** We will also focus on the essential aspects of self-stabilization in distributed contexts

RECAP

What did you learn so far?

- ➔ Challenges in Message Passing systems
- ➔ Distributed Sorting
- ➔ Space-Time Diagram
- ➔ Partial Ordering / Causal Ordering
- ➔ Concurrent Events
- ➔ Local Clocks and Vector Clocks
- ➔ Distributed Snapshots
- ➔ Termination Detection
- ➔ Topology Abstraction and Overlays
- ➔ Leader Election Problem in Rings
- ➔ Message Ordering / Group Communications
- ➔ Distributed Mutual Exclusion Algorithms

Topics to focus on …

- Distributed Mutual Exclusion
- Deadlock Detection
- Check Pointing and Rollback Recovery

- **Self-Stabilization**

- Distributed Consensus

- Peer – to – peer computing and Overlays
- Authentication in Distributed Systems

For End Semester

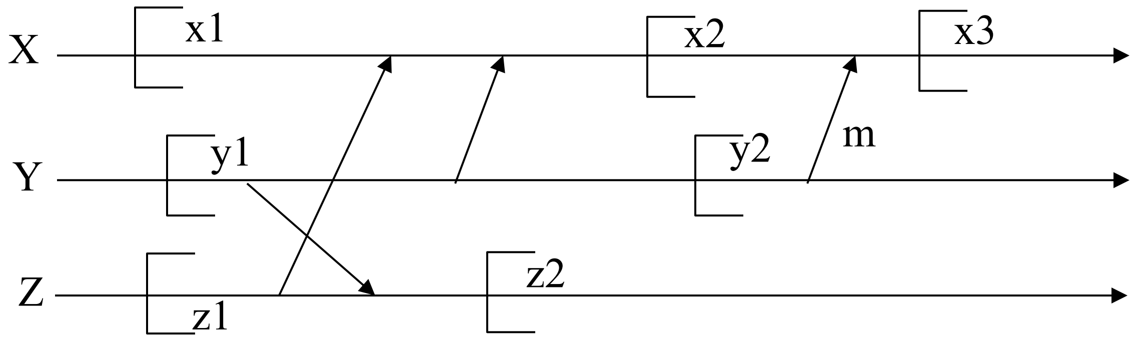
Self-Stabilization in Distributed Systems

Let us explore Self-Stabilization algorithms in Distributed Systems

Handling Failures / Recovery?

- Failure of a site/node in a distributed system causes inconsistencies in the state of the system.
- Recovery: bringing back the failed node in step with other nodes in the system.
- Failures:
  - Process failure:
    - Deadlocks, protection violation, erroneous user input, etc.
  - System failure:
    - Failure of processor/system. System failure can have full/partial amnesia.
    - It can be a pause failure (system restarts at the same state it was in before the crash) or a complete halt.
  - Secondary storage failure: data inaccessible.
  - Communication failure: network inaccessible.

Consistent Checkpoints



- Overcoming domino effect and livelocks: checkpoints should not have messages in transit.
- Consistent checkpoints: no message exchange between any pair of processes in the set as well as outside the set during the interval spanned by checkpoints.
- {x1,y1,z1} is a strongly consistent checkpoint

Types of CP-RR Algorithms

- Synchronous Algorithm
  - Two Phase algorithm proposed by Koo and Toueg
- Asynchronous Algorithm
  - A simple algorithm proposed by Juang & Venkatesan

Overview

- Self-Stabilizing (SS) Systems
  - Legitimate / Illegitimate states
  - System Model
  - Token Ring System
    - Dijkstra's Self-stabilizing Algorithm
    - Construct Breadth-First Trees (BFT)
  - Computational Cost
  - Fault Tolerance / Factors Preventing SS
  - Limitations of SS systems

Introduction

- ➔ Legitimate State – Systems behave correctly as it has expected to.
- ➔ Illegitimate State – inactive state or state in which the system misbehaves (Message is lost)
- ➔ Self – Stabilization – A concept of fault-tolerance in distributed computing
- ➔ Regardless of initial state, system is guaranteed to converge to a legitimate state in a finite amount of time without any outside intervention
- ➔ Problem – Nodes do not have a global memory

Definition

A system is self-stabilizing if and only if:

- ➔ Convergence: Starting from any state, it is guaranteed that the system will eventually reach a correct state
- ➔ Closure: Given that the system is in a correct state, it is guaranteed to stay in a correct state, provided that no fault happens

- ➔ A system is said to be randomized self-stabilizing if and only if it is self-stabilizing and the expected number of rounds needed to reach a correct state is bounded by some constant k

System Model

- An abstract computer model: state machine.
- A distributed system model comprises of a set of n state machines called processors that communicate with each other, which can be represented as a GRAPH
- Message passing communication model:
  - queue(s) Q_{ij}, for messages from P_i to P_j
- System configuration is set of states, and message queues.
- In any case it is assumed that the topology remains connected, i.e., there exists a path between any two nodes.

Token Rings

- Dijkstra's Self-Stabilizing Token Ring System
  - When a machine has a privilege, it is able to change its current state, which is referred to as a move.
  - A legitimate state must satisfy the following constraints:
    - There must be at least one privilege in the system (liveness or no deadlock).
    - Every move from a legal state must again put the system into a legal state (closure).
  - During an infinite execution, each machine should enjoy a privilege an infinite number of times (no starvation)
  - Given any two legal states, there is a series of moves that change one legal state to the other (reachability).
    Dijkstra considered a legitimate (or legal) state as one in which exactly one machine enjoys the privilege

Dijkstra's Algorithm

- For any machine:
  - S – State of its own
  - L – State of the left neighbor and
  - R - State of the right neighbor on the ring

- The exceptional machine:
  - If L = S then S = (S+1) mod K;

- All other machines:
  - If L = S then S = L;

Dijkstra's Algorithm

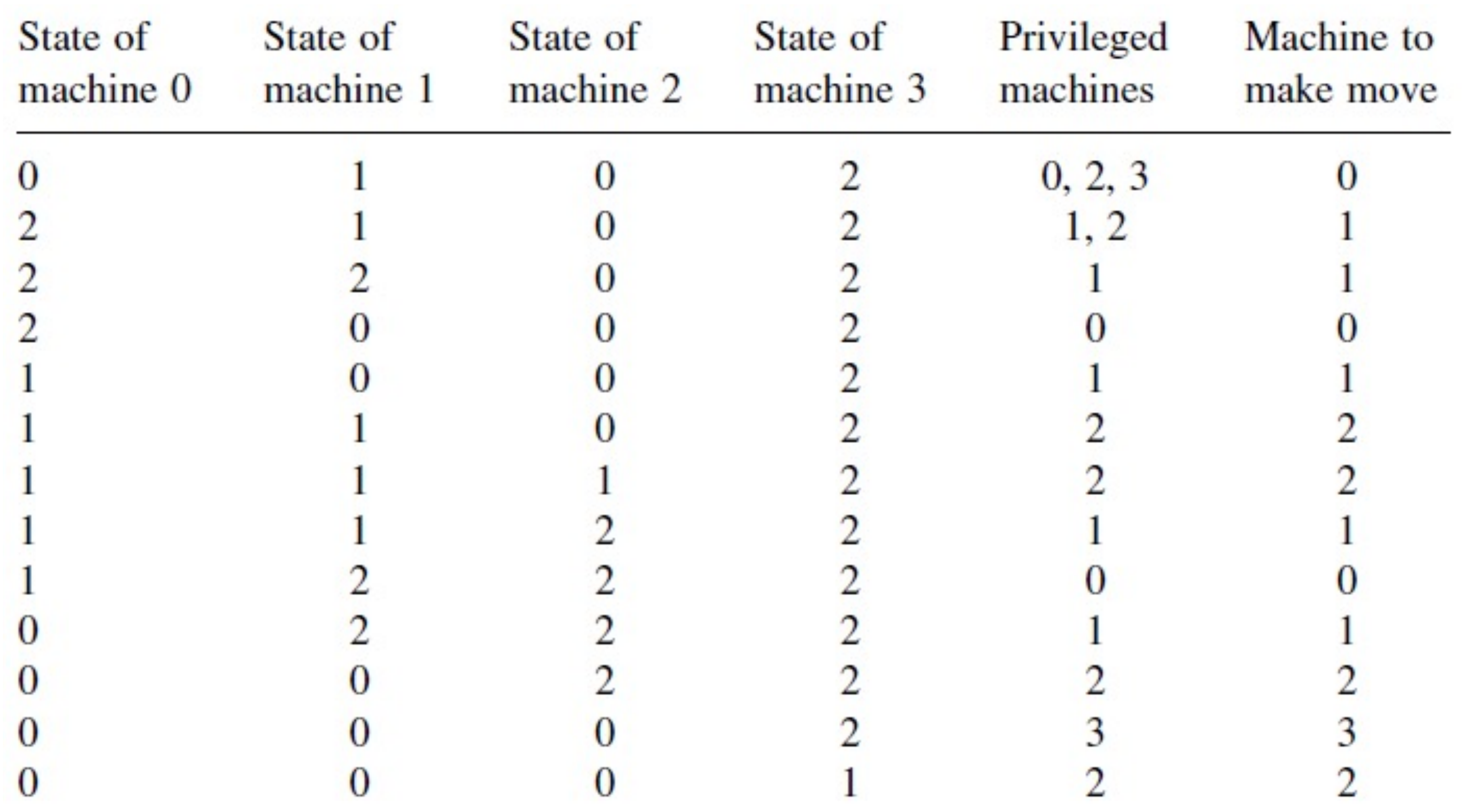
- A Privilege of a machine is able to change its current state on a Boolean predicate that consists of its current state and the states of its neighbors
- When a machine has a privilege, it is able to change its current state, which is referred to as a move.

Second solution (K = 3)

- The bottom machine, machine 0:
  - If (S+1) mod 3 = R then S = (S−1) mod 3;
- The top machine, machine n−1:
  - If L = R and (L+1) mod 3 = S then S = (L+1) mod 3;
- The other machines:
  - If (S+1) mod 3 = L then S = L;

An Illustration

- 4 Machines: M0, M1, M2, and M3



State of machine 0	State of machine 1	State of machine 2	State of machine 3	Privileged machines	Machine to make move
0	1	0	2	0, 2, 3	0
2	1	0	2	1, 2	1
2	2	0	2	1	1
2	0	0	2	0	0
1	0	0	2	1	1
1	1	0	2	2	2
1	1	1	2	2	2
1	1	2	2	1	1
1	2	2	2	0	0
0	2	2	2	1	1
0	0	2	2	2	2
0	0	0	2	3	3
0	0	0	1	2	2

Fault Tolerance

A Self-Stabilizing System handles Transient faults:

- ➔ Inconsistent Initialization: Different processes initialized to local states that are inconsistent with one another.
- ➔ Mode of Change: There can be different modes of execution of a system. In changing the mode of operation, it is impossible for all processes to effect the change in same time.
- ➔ Transmission Errors: Loss, corruption, or reordering of messages
- ➔ Memory Crash

Factors Preventing Self-Stabilization

Transient faults:

- ➔ Symmetry: Processes should not be identical/symmetric because solution generally relies on a distinguished process.
- ➔ Termination: If any unsafe global state is a final state, system will not be able to stabilize
- ➔ Isolation: Inadequate communication among processes can lead to local states consistent, however, the resulting global state is not safe!
- ➔ Look-alike configurations: Such configurations result when the same computation is enabled in two different states with no way to differentiate between them. Then system cannot guarantee convergence from unsafe state

Limitations of Self-Stabilizing

- Need for an exceptional machine

- Convergence-response tradeoffs
  - Convergence span denotes the maximum number of critical transitions made before the system reaches a legal state
  - Response span denotes the maximum number of transitions to get from the starting state to some goal state
  - Critical Transitions. (ex. A process moves into a critical section, while another is already in!)

Limitations of Self-Stabilizing (contd)

- Pseudo-stabilization: Weaker, but less expensive with respect to self-stabilization.
  - Every computation only needs to have some state such that the suffix of the computation beginning at this state is in the set of legal computations.
- Verification of self-stabilizing system
  - Verification may be difficult.
  - Stair method developed; Proving the algorithm stabilizes in each step verifies correctness of the entire algorithm, where interleaving assumptions are relaxed

Costs of Self-Stabilization

- Assessment of cost factor
  - Convergence Span: The maximum number of transitions that can be executed in a system, starting from an arbitrary state, before it reaches a safe state.
  - Response Span: The maximum number of transitions that can be executed in a system to reach a specified target state, starting from some initial state. The choice of initial state and target state depends upon the application

Interesting Algorithms

- ➔ Breadth-First Trees (Huang and Chen, 1992)
- ➔ All-pairs shortest path problem (Chandrasekar and Srimani, 1994)
- ➔ Finding centers and medians of trees (Bruell et al. 1999)
- ➔ Shortest path problem (Huang and Lin, 2002)
- ➔ Shortest path problem assuming read/write atomicity (Huang, 2005)
- ➔ Connected minimal dominating sets (Turau and Hauck, 2009)
- ➔ Finding efficient sets of graphs and trees (Turau, 2013)
- ➔ Leader election (Altisen et al., 2017)
- ➔ Edge monitoring in wireless sensor networks (Neggazi et al., 2017)

Breadth-First Trees

- Proposed by Huang and Chen, 1992
- Breadth-First Tree (BFT): A Breadth-First Tree of a connected graph is a spanning tree of the graph in which each node has a minimum distance to the root along the tree edges
- How to construct a BFT from a given graph?
- How to develop a self-stabilizing algorithm for constructing the Breadth-First Tree?

Self-Stabilizing Algorithm for BFT

- **Basics:**
  - Model a distributed system as a connected graph G(V, E)
  - A specific node r is selected as the root.
  - How to build a breadth-first- tree rooted at r from G with each node knowing its level in the tree.
  - For each node i, let N_i be the set of i's neighbors

  - Each node i other than the root maintains the following two local variables:
    - L(i): the level of i,
    - P(i) : the parent of i,

    where $2 <= L(i) <= n$ and P(i) in N_i

Self-Stabilizing Algorithm for BFT

- From G, construct BFT rooted at node r
- In a tree:
  - $L(i) = (L(p_i) + 1)$ for any i other than r
  - $L(pi) = min(\{L(j) \mid j \ in \ N_j \})$ based on the property of breadth-first trees.

- The system reaches a legitimate state, when the following predicate is true

$$BFT = (\forall i{:}\ i \# r{:}\ L(i) = L(p_i) + 1 \wedge L(p_i) = min(\{L(j) \mid j \ in \ N_i\}))$$

Self-Stabilizing Algorithm for BFT

- For any node i, if $L(i) <= L(p_i)$, we call node i an L-*turn* node, or more specifically a k-*turn* node, where $k = L(i)$. Also let t_k be the number of all the k-*turn* nodes in the system
- Define F_1 as follows:

$$F_1 \equiv (t_2, t_3, \ldots, t_n)$$

- Compare the values of F_1 is by lexicographical order
- Based on lexicographical order,

$(a_1, a_2, \ldots) > (b_1, b_2, \ldots)$ if there exists some k such that $a_i = b_i$, $1 <= i < k$, and $a_k > b_k$

Self-Stabilizing Algorithm for BFT

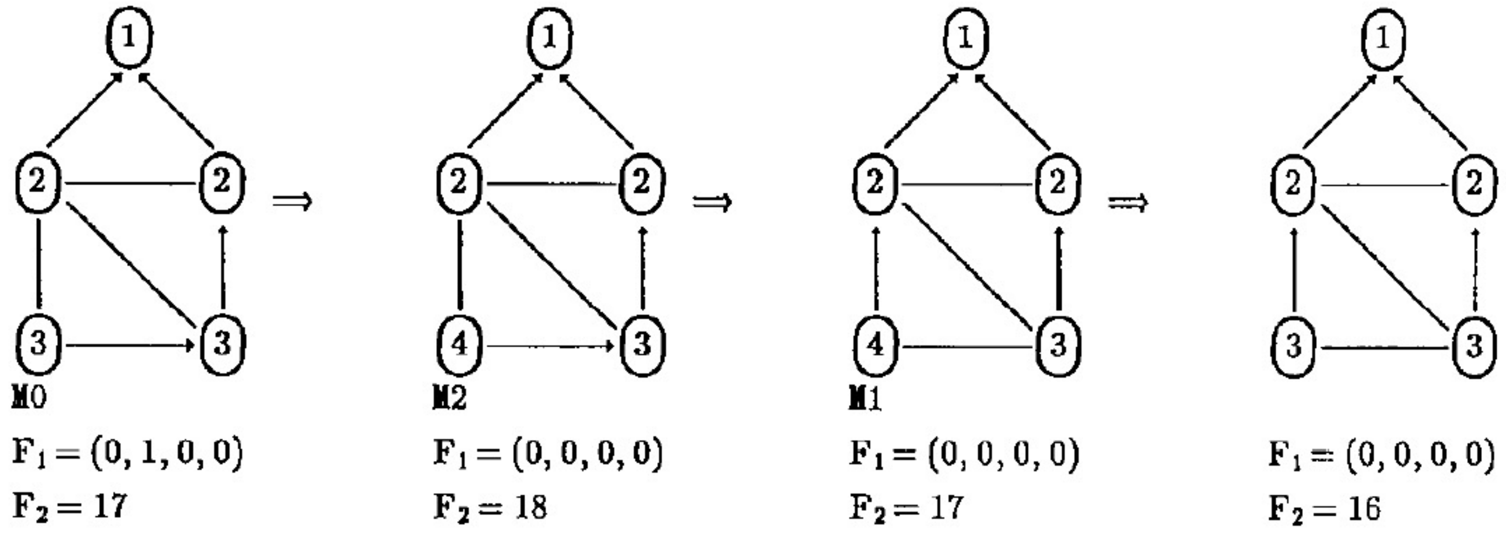
- Define F_2 as follows:

$$F_2 \equiv \sum_{i,\; i \# r} \left(L(i) + L(p_i) \right)$$

- That is, for each node i other than the root, it contributes two values to F_2: one is the level of itself and the other is the level of its parent

An Example

- Here is an illustration



Summary

- ➔ State of Machines
- ➔ Legitimate / Illegitimate States

- ➔ Self-Stabilizing Algorithms
  - ➔ Dijkstra's algorithm (token rings)
  - ➔ Constructing a Breadth First Tree

- ➔ Fault Tolerance
- ➔ Costs of self-stabilization

➔ Stay tuned … More to come up … !!

How to reach me?

- ➔ Please leave me an email:

  rajendra [DOT] prasath [AT] iiits [DOT] in

- ➔ Visit my homepage @

  - ➔ http://www.iiits.ac.in/FacPages/index-rajendra.html

  OR

  - ➔ http://rajendra.2power3.com

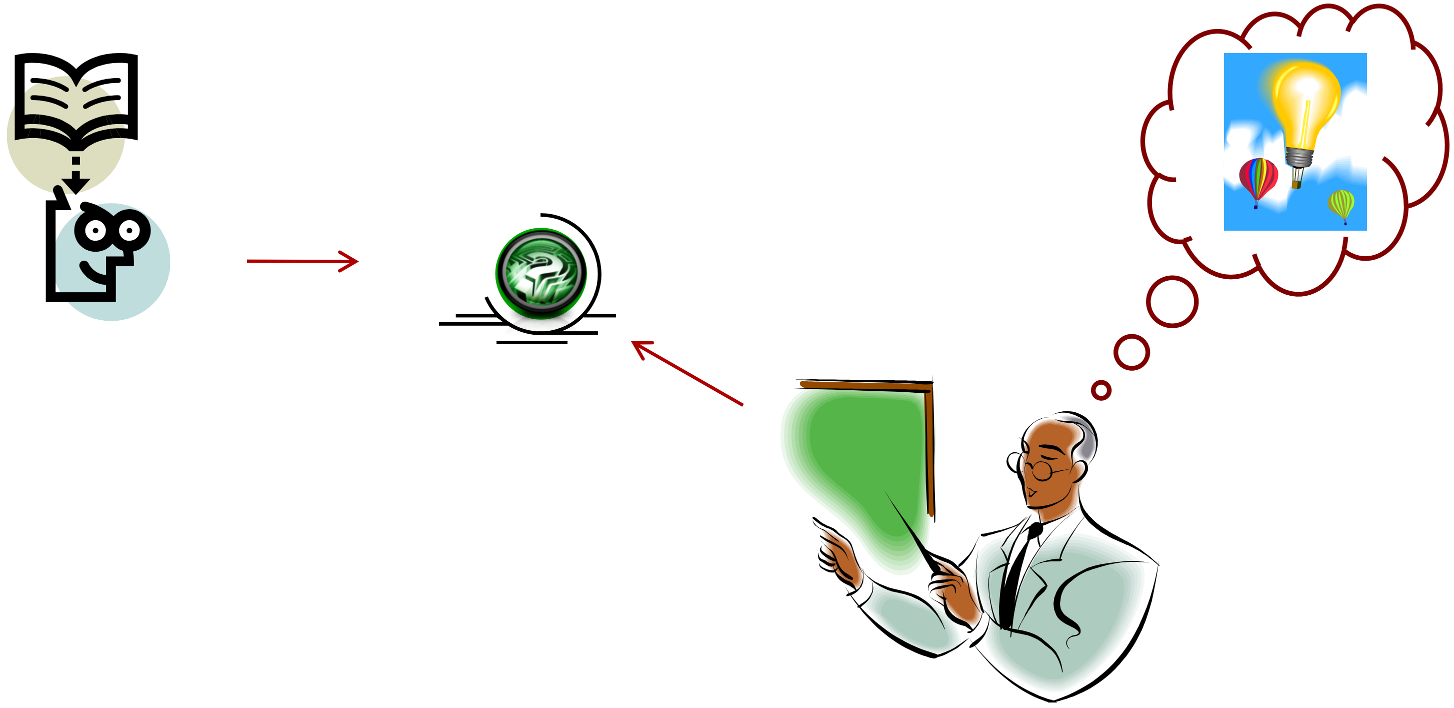
Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Thanks ...



... Questions ???