Distributed Synchronization

EECS 498
Farnam Jahanian
University of Michigan

Reading List

- Tanenbaum Chapter 5.1, 5.4 and 5.5
- Clock Synchronization (optional)
- Distributed Election
- Mutual Exclusion
Clock Synchronization

When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Physical Clocks (1)

Computation of the mean solar day.
Physical Clocks (2)

TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.

Clock Synchronization Algorithms

The relation between clock time and UTC when clocks tick at different rates.
Cristian's Algorithm

Both $T_0$ and $T_1$ are measured with the same clock

The Berkeley Algorithm

a) The time daemon asks all the other machines for their clock values
b) The machines answer
c) The time daemon tells everyone how to adjust their clock

Getting the current time from a time server.
Leader Election Algorithm

• *Common synchronization problem:* Election of one process as a coordinator, initiator or sequencer

• Is a completely symmetric solution possible?
  – NO! Each process must have a unique id.

• *Goal:* Given a set of processes each with a unique id, the goal of an election algorithm is to ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be.

The Bully Algorithm - Garcia-Molina 1982

• When a process notices that the coordinator is no longer around, it initiates an election.

• A process P executes these steps:
  1. P sends an ELECTION message to all processes with higher number.
  2. If no one responds, P wins the election and becomes coordinator.
  3. If one of the processes with higher id answers OK, P’s job is done.

• If a process Q receives an ELECTION message from a lower numbered process id:
  1. Q sends OK back to sender.
  2. Q holds an election, unless it is already holding one.

• Eventually, all processes give up except the one with highest id. It sends a message COORDINATOR to all processes announcing that it is the coordinator.
The Bully Algorithm

(a) Process 4 holds an election
(b) Process 5 and 6 respond, telling 4 to stop
(c) Now 5 and 6 each hold an election

(d) Process 6 tells 5 to stop
(e) Process 6 wins and tells everyone
The Bully Algorithm

• What are the assumptions for the Bully Algorithm?
  – Crash failures
  – Synchronous processes
  – Reliable message delivery
  – Timeout on message delivery (response) indicates process crash → synchronous communication (bounded message delay)

• What happens when a new process with higher id restarts?
  – It initiates a new election algorithm.

A Ring Algorithm

• Logically order processes in a ring such that each process knows who its successor is.

• When a process notices that the coordinator is not functioning, it sends an ELECTION message with its process id to its successor.

• If the successor is down (???), the sender skips on to the next process.

• At each step, the sender adds its process id to the message.

• Eventually, the message gets back to the initiating process. At that point, the message type is changed to COORDINATOR and sent around once again to let everyone know who the coordinator is.

• When the COORDINATOR message goes around once, it is removed by the initiator.
A Ring Algorithm

Election algorithm using a ring.

A Ring Algorithm

• What if two processes discover almost simultaneously... and start the election algorithm by circulating the ELECTION message?

• Assumptions for this protocol?
Mutual Exclusion

Motivation: How to ensure mutual exclusive access to a critical region that is protecting a shared resource or data structure in a distributed system?

a) Centralized algorithm
b) Distributed algorithm
c) Token-based algorithm

Mutual Exclusion
A Centralized Algorithm

Implement it using a centralized lock manager.

Request/release msgs. are sent to a central coordinator.

When the lock is granted and a new request comes in, it may be queued or denied.

Advantages:
Correct ... simple ... free from starvation ... free from deadlock

Disadvantages:
Single point of failure ... performance bottleneck ...
Mutual Exclusion:
A Centralized Algorithm

a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted.
b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
c) When process 1 exits the critical region, it tells the coordinator, when then replies to 2.

Mutual Exclusion:
A Distributed Algorithm

First introduced by Lamport ... later optimized by others.
Assume total ordering of all events in the system.
Assume reliable message delivery with ack msgs.

A requesting process adds its process id and current time to the REQUEST message and sends it to all processes.

When a process receives a REQUEST message:
– If receiver process is not in the critical region, ack with OK.
– If it is in the critical region, send no reply. Queue request.
– If receiver is waiting to enter critical region, compare the timestamp of incoming msg. with own request timestamp. The lower timestamp wins. If incoming message timestamp lower, send OK back to sender. If own timestamp lower, send no reply ... queue message.

After receiving OK from everyone, enter critical region.
Upon exit from a critical region, send OK to all waiting (requests queued locally)... Empty queue.
A Distributed Algorithm

a) Two processes want to enter the same critical region at the same moment.
b) Process 0 has the lowest timestamp, so it wins.
c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

Mutual Exclusion: A Distributed Algorithm

Advantages:
- Free from starvation and deadlock ... no single point of failure!

Disadvantages:
- n points of failure! A single process failure can block the system if it fails after queuing a REQUEST.
- One possible solution is to reply with DENIED msg. and queue REQUEST.
- Some form of group membership protocol is needed anyway to keep track of joins, departures and crashes.
- ALL processes are involved in ALL decisions! Won't scale ...
- Majority response is sufficient!
- Slower ... more complicated ... more prone to failures ... won't scale ... but it is distributed.
Mutual Exclusion:  
A Token Ring Algorithm

- Assume a bus network.
- Arrange processes in a logical ring.
- When ring is initialized, process 0 has the token.
- Token goes around the ring from process k to process k+1.
- A process can enter the critical region if it has the token.
- Upon exiting from the critical region, the token is passed.
- Token just circulates if no process needs it.

- Advantages:
  - free from deadlock and starvation ... correct ... relatively simple ...

- Potential complications:
  - handling token loss ... token loss vs. a very long critical region.
  - handling process crash ... maintain ring membership.

- Assignment: Propose a token loss detection algorithm.

---

A Token Ring Algorithm

(a) An unordered group of processes on a network.
(b) A logical ring constructed in software.
## Comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Messages per entry/exit</th>
<th>Delay before entry (in message times)</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>3</td>
<td>2</td>
<td>Coordinator crash</td>
</tr>
<tr>
<td>Distributed</td>
<td>2 ((n - 1))</td>
<td>2 ((n - 1))</td>
<td>Crash of any process</td>
</tr>
<tr>
<td>Token ring</td>
<td>1 to (\infty)</td>
<td>0 to (n - 1)</td>
<td>Lost token, process crash</td>
</tr>
</tbody>
</table>

A comparison of three mutual exclusion algorithms.