Chapter 6
Synchronization

Physical Clocks (1)

Figure 6-2. Computation of the mean solar day.

Physical Clocks (2)

Figure 6-3. TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.
Global Positioning System (1)

Figure 6-4. Computing a position in a two-dimensional space.

Global Positioning System (2)

Real world facts that complicate GPS

1. It takes a while before data on a satellite’s position reaches the receiver.
2. The receiver’s clock is generally not in sync with that of a satellite.

Clock Synchronization Algorithms

Figure 6-5. The relation between clock time and UTC when clocks tick at different rates.

Network Time Protocol

Figure 6-6. Getting the current time from a time server.
The Berkeley Algorithm (1)

Figure 6-7. (a) The time daemon asks all the other machines for their clock values.

The Berkeley Algorithm (2)

The Berkeley Algorithm (3)

Clock Synchronization in Wireless Networks (1)

Figure 6-8. (a) The usual critical path in determining network delays.
Lamport's Logical Clocks (1)

The "happens-before" relation \( \rightarrow \) can be observed directly in two situations:

- If \( a \) and \( b \) are events in the same process, and \( a \) occurs before \( b \), then \( a \rightarrow b \) is true.
- If \( a \) is the event of a message being sent by one process, and \( b \) is the event of the message being received by another process, then \( a \rightarrow b \)

Lamport's Logical Clocks (2)

Figure 6-9. (a) Three processes, each with its own clock. The clocks run at different rates.

Lamport's Logical Clocks (3)

Figure 6-9. (b) Lamport's algorithm corrects the clocks.
Lamport's Logical Clocks (4)

Figure 6-10. The positioning of Lamport's logical clocks in distributed systems.

Example: Totally Ordered Multicasting

Figure 6-11. Updating a replicated database and leaving it in an inconsistent state.

Lamport's Logical Clocks (5)

Updating counter $C_i$ for process $P_i$

1. Before executing an event $P_i$, executes $C_i = C_i + 1$.
2. When process $P_i$ sends a message $m$ to $P_j$, it sets $m$'s timestamp $ts(m)$ equal to $C_i$ after having executed the previous step.
3. Upon the receipt of a message $m$, process $P_j$ adjusts its own local counter as $C_j = \max(C_j, ts(m))$, after which it then executes the first step and delivers the message to the application.

Vector Clocks (1)

Figure 6-12. Concurrent message transmission using logical clocks.
Vector Clocks (2)

Vector clocks are constructed by letting each process $P_i$ maintain a vector $VC_i$, with the following two properties:

1. $VC_i[j]$ is the number of events that have occurred so far at $P_i$. In other words, $VC_i[j]$ is the local logical clock at process $P_i$.
2. If $VC_i[j] = k$ then $P_i$ knows that $k$ events have occurred at $P_i$. It is thus $P_i$'s knowledge of the local time at $P_i$.

Vector Clocks (3)

Steps carried out to accomplish property 2 of previous slide:

1. Before executing an event $P_i$ executes $VC_i[i] = VC_i[i] + 1$.
2. When process $P_i$ sends a message $m$ to $P_j$, it sets $m$'s (vector) timestamp $ts(m)$ equal to $VC_i$ after having executed the previous step.
3. Upon the receipt of a message $m$, process $P_j$ adjusts its own vector by setting $VC_j[k] = \max(VC_j[k], ts(m)[k])$ for each $k$, after which it executes the first step and delivers the message to the application.

Enforcing Causal Communication

![Diagram showing causal communication](image)

Figure 6-13. Enforcing causal communication.

Mutual Exclusion

A Centralized Algorithm (1)

![Diagram showing mutual exclusion](image)

(a) Process 1 asks the coordinator for permission to access a shared resource. Permission is granted.
A Distributed Algorithm (1)

Three different cases:
1. If the receiver is not accessing the resource and does not want to access it, it sends back an OK message to the sender.
2. If the receiver already has access to the resource, it simply does not reply. Instead, it queues the request.
3. If the receiver wants to access the resource as well but has not yet done so, it compares the timestamp of the incoming message with the one contained in the message that it has sent everyone. The lowest one wins.

A Distributed Algorithm (2)

Figure 6-15. (a) Two processes want to access a shared resource at the same moment. (b) Accesses resource
**A Distributed Algorithm (3)**

Figure 6-15. (b) Process 0 has the lowest timestamp, so it wins.

**A Distributed Algorithm (4)**

Figure 6-15. (c) When process 0 is done, it sends an OK also, so 2 can now go ahead.

**A Token Ring Algorithm**

Figure 6-16. (a) An unordered group of processes on a network. (b) A logical ring constructed in software.

**A Comparison of the Four Algorithms**

<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>Decentralized</td>
<td>3mk, k = 1,2,...</td>
<td>2 m</td>
<td>Starvation, low efficiency</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 ∞</td>
<td>0 to n − 1</td>
<td>Lost token, process crash</td>
</tr>
</tbody>
</table>

Figure 6-17. A comparison of three mutual exclusion algorithms.
Election Algorithms

The Bully Algorithm
1. P sends an ELECTION message to all processes with higher numbers.
2. If no one responds, P wins the election and becomes coordinator.
3. If one of the higher-ups answers, it takes over. P’s job is done.
Figure 6-20. The bully election algorithm. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

A Ring Algorithm

Figure 6-21. Election algorithm using a ring.

Elections in Wireless Environments (1)

Elections in Wireless Environments (2)
Elections in Large-Scale Systems (1)

Requirements for superpeer selection:
1. Normal nodes should have low-latency access to superpeers.
2. Superpeers should be evenly distributed across the overlay network.
3. There should be a predefined portion of superpeers relative to the total number of nodes in the overlay network.
4. Each superpeer should not need to serve more than a fixed number of normal nodes.

Elections in Large-Scale Systems (2)

Figure 6-23. Moving tokens in a two-dimensional space using repulsion forces.