Global State Collection
Global state collection

Some applications

- computing network topology
- termination detection
- deadlock detection

Chandy-Lamport algorithm does a partial job. Each process generates a fragment of the global state, but these pieces have to be “stitched together” to form a global state.
A simple exercise

Once the pieces of a consistent global state become available, consider collecting the global state via *all-to-all broadcast*

At the end, each process will compute a set $V$, where

$$V = \{ s(i) : 0 \leq i \leq N-1 \}$$
All-to-all broadcast

Assume that the topology is a strongly connected graph

program broadcast {for process i}
define $V_i, W_i$: set of values;
initially $V_i = \{s(i)\}, W_i = \emptyset$ {and every channel is empty}

1 do $V_i \neq W_i$ → send $(V_i \setminus W_i)$ to every outgoing channel;
   $W_i := V_i$
2 [] $\neg empty(k,i)$ → receive X from channel $(k,i)$;
   $V_i := V_i \cup X$

od

Acts like a “pump”
**Proof outline**

**Lemma.** empty \((i, k) \Rightarrow W.i \subseteq V.k.\)

Use a simple inductive argument.

**(Base case)** \(W.i = \text{null}\)

**(Induction hypothesis)** Assume it is true now.

**(Inductive step)** Show that when \((i,k)\) becomes empty next time, the condition holds. Easy!

**(Upon termination)** \(\forall i: V.i = W.i,\)

and all channels are empty. So, \(V.i \subseteq V.k.\)

In a strongly connected graph, every node is in a cycle. On a cyclic path, \(V.i = V.k\) must be true. This means \(\forall i, j: V.i = V.j\) since \(s(i) \in V.i,\)

\(\forall j: s(i) \in V.j.\)
Proof outline

**Lemma.** The algorithm will terminate in a bounded number of steps.

Consider the variant function

\[ Y = (V_0, V_1, V_2, ..., V_{n-1}, c_0, c_1, c_2, ..., c_{m-1}) \]

Statement 1 and Statement 2 increase \( Y \) lexicographically, until the \( V \)'s reach their largest value \( \{s(0), s(1), s(2), \ldots, s(n-1)\} \).

So the algorithm terminates in a bounded number of steps.
Termination detection

During the progress of a distributed computation, processes may periodically turn active or passive.

A distributed computation terminates when:

(a) every process is passive,
(b) all channels are empty, and
(c) the global state satisfies the desired postcondition.
Visualizing diffusing computation

Notice how one process engages another process. Eventually all processes turn white, and no message is in transit -- this signals termination.

How to develop a signaling mechanism to detect termination?
Dijkstra-Scholten algorithm

The basic scheme

An initiator initiates termination detection by sending signals (messages) down the edges via which it engages other nodes.

At a “suitable time,” the recipient sends an ack back.

When the initiator receives ack from every node that it engaged, it detects termination.
Dijkstra-Scholten algorithm

Deficit (e) = # of signals on edge e - # of acks on edge e

For any node, C = total deficit along incoming edges
and D = total deficit along outgoing edges

For the initiator, by definition, C = 0

Dijkstra-Scholten algorithm used the following two invariants to develop their algorithm:

Invariant 1. (C ≥ 0) ∧ (D ≥ 0) (obvious, deficit ≥ 0)
Invariant 2. (C > 0) ∨ (D = 0) (proposed invariant)

To be observed by the signaling scheme
Dijkstra-Scholten algorithm

The invariants must hold when an interim node sends an ack. So, acks will be sent when

\[(C - 1 \geq 0) \land (C - 1 > 0 \lor D = 0)\]

{follows from INV1 and INV2}

\[= (C > 1) \lor (C \geq 1 \land D = 0)\]

\[= (C > 1) \lor (C = 1 \land D = 0)\]
Dijkstra-Scholten algorithm

program detect \{for an internal node \( i \)\}

\textbf{define} \quad C, D : \text{integer} \\
\quad m : \text{(signal, ack)} \quad \{\text{represents the type of message received}\} \\
\quad state : \text{(active, passive)} \\
\textbf{initially} \quad C = 0, \quad D = 0, \quad \text{parent}(i) = i \\

\textbf{do} \quad (m = \text{signal}) \land (C = 0) \quad \rightarrow \quad C : = 1; \quad state : = \text{active}; \\
\quad \quad \quad \text{parent} : = \text{sender} \\
\quad \{\text{Read Note 1}\} \\
\text{[] (m = \text{ack})} \quad \rightarrow \quad D : = D - 1 \\
\text{[] (C = 1 \land D = 0) \land (state = \text{passive})} \quad \rightarrow \quad \text{send ack to parent}; \\
\quad \quad \quad C : = 0; \quad \text{parent}(i) = i \\
\quad \{\text{Read Note 2}\} \\
\textbf{[] (m = \text{signal}) \land (C = 1)} \quad \rightarrow \quad \text{send ack to the sender} \\
\textbf{od}

\textbf{Note 1.} The node will forward signals to its successors (whenever they exist) \\
\textbf{Note 2.} The node now disappears from the computation graph. \\
The \textit{parent} relation induces a spanning tree
Distributed deadlock

When each process waits for some other process (to do something), a deadlock occurs.

Assume each process owns a few resources. Review how resources are allocated, and how a deadlock is created.

*Three criteria for the occurrence of deadlock*

- Exclusive use of resources
- Non-preemptive scheduling
- Circular waiting by all (or a subset of) processes
Distributed deadlock

*Three aspects of deadlock*

- deadlock detection
- deadlock prevention
- deadlock recovery
Distributed deadlock

• May occur due to bad designs/bad strategy

[Sometimes prevention is more expensive than detection and recovery. So designs may not care about deadlocks, particularly if it is rare.]

• Caused by failures or perturbations in the system
Distributed Deadlock Prevention uses pessimistic strategies

An example from banker’s problem (Dijkstra)

A banker has $10,000. She approves a credit line of $6,000 to each of the three customers A, B, C. since the requirement of each is less than the available funds.

1. The customers can pay back any portion of their loans at any time. Note that no one is required to pay any part of the loan unless (s)he has borrowed up to the entire credit line.

2. However, after the customer has borrowed up to the entire credit line ($6000) (s)he must return the entire money is a finite time.

3. Now, assume that A, B, C borrowed $3000 each.

The state is unsafe since there is a “potential for deadlock.” Why?
Banker’s Problem

Questions for the banker

Let the current allocations be $A = 2000, \ B = 2400, \ C = 1800.$

1. Now, if $A$ asks for an additional $1500, then will the banker give the money immediately?

2. Instead, if $B$ asks for $1500 then will the banker give the money immediately?
Wait-for Graph (WFG)

- Represents who waits for whom.
- No single process can see the WFG.
- Review how the WFG is formed.
Another classification

• **Resource deadlock**
  \[R1 \text{ AND } R2 \text{ AND } R3 \ldots\]
  also known as AND deadlock

• **Communication deadlock**
  \[R1 \text{ OR } R2 \text{ OR } R3 \ldots\]
  also known as OR deadlock
Detection of resource deadlock

Notations

\[ w(j) = \text{true} \Rightarrow (j \text{ is waiting}) \]

\[ \text{depend } [j,i] = \text{true} \Rightarrow j \in \text{succ}^n(i) \text{ (n>0)} \]

(i’s progress depends on j’s progress)

\[ P(i,s,k) \text{ is a probe} \]

(i=initiator, s= sender, k3=receiver)
Detection of resource deadlock

{Program for process k}

\[\textbf{do} \quad P(i,s,k) \text{ received } \wedge \]
\[w[k] \wedge (k \neq i) \wedge \neg \text{depend}[k, i] \rightarrow \]
send \(P(i,k,j)\) to each successor \(j\);
\text{depend}[k, i] := \text{true}

\[\textbf{od} \]

\[\text{process k is deadlocked}\]
Observations

To detect deadlock, the initiator must be in a cycle

Message complexity = $O(|E|)$

(edge-chasing algorithm)
This WFG has a resource deadlock but no communication deadlock
Detection of communication deadlock

A process ignores a probe, if it is not waiting for any process. Otherwise,

- **first** probe →
  - mark the sender as *parent*;
  - forwards the probe to successors
- Not the first probe →
  - Send ack to that sender
- **ack** received from every successor →
  - send ack to the parent

Communication deadlock is detected if the initiator receives **ack**.

*Has many similarities with Dijkstra-Scholten’s termination detection algorithm*