Distributed Deadlock Detection

by M. Chandy, J. Misra, and L. Haas, ACM TOCS, 1983

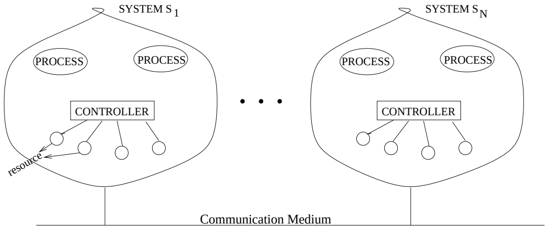
- Ways to handle deadlock
  - avoidance
  - prevention
  - detection $\rightarrow$ *this paper*
- Deadlock types
  - Resource deadlock
    (circular wait)
  - Communication deadlock
    (choice of multiple resources)
  - Harder!

Related Work

- Dijkstra and Scholten
- Termination detection
- "Diffusing computation model"
- Differences
  - CSP-like model
  - wait selecting messages from *some* not necessarily *all* processes
  - any message may restart the waiting process
  - detect deadlock in *subsets* of processes
  - in contrast to waiting for all processes

Resource Deadlock

- Common in distributed database systems...
- Mental Model



- Controller controls resources

- Controller responsibilities
  - process scheduling
  - resource management
  - communication
- Transaction = $\{P_{a1}, ..., P_{aN}\}$
- If P_{ij} wants a resource

  P_{ij} sends a request to C_j (local controller)

  If resource is local+free, C_j allocates it to P_{ij}

  If resource is non-local but controlled by C_m then:

  1. C_j sends mesg to P_{im} (the counter part of your transaction to system m) via C_m
  2. P_{im} requests resource (and gets it) from C_m
  3. P_{im} then communicates this via C_j back to P_{ij} via C_m.

Communication Model

- Each transaction by itself is OK and won't deadlock
- Communication medium is FIFO and reliable
- Each idle process has its own dependent processes (and knows which ones they are!)
- Messages are sent from A to B and are expected to be received in the order sent
- *Definition:* A set of processes is deadlocked iff
  - all processes in S are idle
    (idle $\Rightarrow$ non-empty set of dependent), otherwise terminated
  - each of these is in S
  - no communication in progress

Types of Deadlock Models

Resource Deadlocks	Communication Deadlocks
$\downarrow$	$\downarrow$
serially reusable resources	consumable resources
(physical things:	(non-physical things:
printer, disk, etc)	msgs, perm., semaphores)
|	|
shared memory	message passing
|	|
you know <u>what</u>	you know <u>who</u>
but not <u>who</u>	but not <u>what</u>
you are waiting for	you are waiting for
|	|
wait till all resources	can be released by one
are available	of several

Basic Idea (Resource Model)

- Use concept of probe
- (i, j, k)
  send probe from P_j to P_k concerning P_i
- P_k accepts the probe iff
  - P_k is idle
  - P_k didn't know P_i was dependent on P_k
  - P_k now does know that P_i is dependent on it.
  - C_k isn't sending stuff to C_j
- Every so often, a controller initiates a probe on each of is idle processes

Resource Model – Algorithm

- Data structure: boolean table, if ki element is TRUE, then P_i is dependent on P_k
- Check column i, looking for a TRUE, say it is in row a, P_a
- Is P_a idle? If not, wait (P_i should still be alive...)
- If P_a is idle, scan P_a's column for TRUE, say P_b
- Send probe (i, a, b)
- If process P_k gets probe
  e.g., probe (i, j, k)

Resource Model – Algorithm (cont.)

- When process P_k gets probe ...
- If P_k accepts it, then
  - $dependent_k(i)$ is set to TRUE
  - if $k = i$, then DEADLOCK
  - pass probe on to P_k's dependents (i.e., scanning column k for TRUE)
- Set column k to FALSE if P_k starts executing
- When P_k goes idle, waiting on a resource held by P_i, then dependent $i(k)$ is set to TRUE
- Deadlock: if P_i accepts probe (i,j,i)
- Basically, we are tracing a dependency tree...

Communication Model

- Each idle process has a set of "dependent processes"
- An idle process starts executing when it receives a message from *any* process in its dependent set, otherwise it does not change state or its dependent set...
- A process is *terminated* if it is idle and its dependent set is empty
- Uses concept of "query computation" when a process does idle. Basically, when process p_i goes idle you send out a bunch of queries (one to each of its dependent set processes).
- If you get replies back from each $\Leftrightarrow$ you are deadlocked, i.e., found a knot

Communication Model (cont.)

- To determine if a process is deadlocked, it initiates a "query computation" when it enters the idle state
- **Primitives:**
  - query(i,m,j,k)
  - reply(i,m,j,k)
  - i: initiating process
  - m: sequence #
  - j: sending process
  - k: receiving process
- If P_i is deadlocked when initiates its mth query computation, it will receive reply (i, m, j, i) corresponding to every query (i, m, i, j) that is sent
- If P_i receives reply (i, m, j, i) corresponding to every query (i, m, i, j) that is sent, then it is deadlocked...

Data Structure for Communication Model

For P_K, largest engaged number

...	process i	...
...	○	←largest seq# seen by P_K with i as initiator **latest(i)**
...	○	← the sender of the above (who I heard sequence # from) **engager(i)**
...	○	# of dependents **num(i)**
...	○	TRUE $\Leftrightarrow$ P_K continually idle since sending **wait(i)**

Algorithm for Communication Model

- Every idle process sends queries to its dependent set
- Suppose P_k received a query or reply with initiator i, sequence # m, and sender j.
- If $m < latest(i)$, P_k discards the message
- If $wait(i)$ is FALSE and $m = latest(i)$, then P_i is not deadlocked
- If $m > latest(i)$, then P_k must set $latest(i)$ to m, $engager(i)$ to j, and $wait(i)$ to TRUE
- If P_k receives a reply with $m = latest(i)$ and $wait(i)$ is TRUE, then it should reduce the number of its dependents by 1.
- If P_k receives a query with $m > latest(i)$, it propagates the query to all processes in its dependent set, updates relevant variables

Algorithm for Communication Model (cont.)

Replies are sent by P_k when

- If $wait(i)$ is TRUE and $m = latest(i)$, replies.
- If $wait(i)$ is FALSE when P_k receives a query with $m = latest(i)$, or $wait(i)$ is TRUE but $m < latest(i)$, does not reply
- If $m > latest(i)$, P_k replies only if $wait(i)$ is TRUE and number of dependents is 0 $\rightarrow P_k$ has received replies to all queries and has been idle since, then initiator is deadlocked

Example...

Recap

- Two algorithms for deadlock detection
- Resource
- Communication
- Correctness is proved
- Efficient algorithms...