Chapter 7

- System Model – typical assumptions underlying the study of distributed deadlock detection
  - Only reusable resources, only exclusive access, single copy of resource in system.
- Besides deadlocks due to resources, we can have communication deadlocks
- Strategies
  - Prevention – can cause further problems. Consider one shot allocation where resources and requesters are on different sites.
  - Avoidance – global state needs to be maintained, safe state checks need to be mutually exclusive.

Detection

- Issues
  - Detection
    - Deadlocks should be detected in finite time.
    - No false positives (phantom deadlocks)
      - Realize that states are not coherent.
  - Resolution
    - Clean up the information upon rollback.
- Control Organizations
    - Centralized
    - Distributed
    - Hierarchical

Distributed Deadlock Detection

- Basic Centralized.
  - All requests and release messages are sent to a designated site, which maintains a global WF Graph
  - Problems – bottleneck, single POF, phantom deadlocks
- Ho-Ramamoorthy 2 Phase
  - Each site maintains a status table – resources locked and resources being waited upon. The central site periodically requests this table, constructs a global WFG, and searches for cycles. If a cycle is found, it requests the tables again, and constructs WFG from those transactions that are common to both tables. If a cycle is still detected, then a deadlock is declared.

- Ho-Ramamoorthy 1 phase
    - Each site maintains 2 tables, one for resources (transactions that have locked a resource) and one for process status (resources locked/waited). These tables requested periodially by central site, and WFG constructed using those entries in the resource table which have corresponding entry in process table.

- Distributed Algorithms
    - Path pushing: WFG constructed by disseminating dependency sequences
    - Edge chasing: process sends out probes. A blocked process receiving probes circulates it along its outgoing dependency edges
    - Diffusion: queries are diffused (sucessively propagated) and reflected
    - Global State Detection

Obermack's Algo.

- Path pushing approach deals with transactions. Each transaction may have sub transactions, but they execute sequentially. Transactions are totally ordered.
- Each site waits for deadlock related information (paths) from other sites. It abstracts the nonlocal portion of the WFG with a single node called EX.
- It combines this with its own WFG. It then detects cycles and breaks those which do not contain EX.
- For all cycles involving EX, the string indicating the cycle (EX-T1-T2-EX) is sent to all sites which have subtransactions of T2 waiting to recv a message from the subtransaction of T2 at this site.
- Problem – this algorithm can detect phantom deadlocks. Needs n(n-1)/2 messages of O(n) size and detects in linear time.

Chandy-Misra- Haas

- Edge chasing algorithm based on the AND model.
- A process Pj is dependent on Pk if there is a sequence Pj, Pi1….Pin,Pk such that all process but Pk are blocked, and each process except Pj has something that is needed by its predecessor.
  - Locally dependent
- If Pi is locally dependent on itself, then we have a deadlock. Otherwise
  - Forall Pj, Pk such that Pi locally depends on Pj and Pj is waiting(not locally) on Pk, send probe(i,j,k) to Pk.

- On receiving probe(i,j,k)
  - If (*Pk is deadlocked* && *! dependent$_k$(i)* && *Pk has not replied to all requests of Pj*)
    - Dependent$_k$(i) = true.
    - If (k == i)
      - » **Then** Pi is deadlocked
      - » **Else** Forall Pm, Pn such that Pk locally depends on Pm and Pm is dependent (not locally) on Pn, send probe(i,m,n) to Pk.
- Sends 1 proble message on each edge of WFG, so m(n-1)/2 messages for a deadlock with m processes over n sites. Size is fixed, and detection time is linear in number of sites

Diffusion Based Algorithm

- Works for OR request model
- Initiation:
  - A blocked process i sends *query(i,i,j)* to all Pj in its dependent set; $num_i(i)$ = |DSi| , $wait_i(i)$ = true;
- When a blocked process Pk recvs query (i,j,k)
  - If this is engaging query, send query(i,k,m) to all processes in its dependent set, and set $num_k(i)$ and $wait_k(i)$
  - Else if $wait_k(i)$ then send reply(i,k,j)
- When Pk gets reply(i,j,k)
  - If $wait_k(i)$
    - Decrement $num_k(i)$, if it becomes 0 then
      - » If k == I then deadlock else reply(i,k,m) to the process which sent the engaging query.

Heirarchical Algorithms

- Menasce-Muntz
  - Resources are managed by nodes that form the “leaves” of a tree. They maintain TWF/WFGs corresponding to the resources they manage.
  - Several leaf controllers have a single parent, and so on in a tree fashion. Each non-leaf controller maintains WFG which is union of child WFGs. Changes are propagated upwards, and deadlocks detected on the way
- Hierarchical Ho-Ramamoorthy
  - Sites split into disjoint clusters.
  - Each cluster has its own control site. There is also a central control site.

Issues

- Formal methods to prove correctness
- Performance metrics
  - No of messages ? Message size? Time to detect ? Storage overhead ? Computation overhead ?
- Resolution – basically aborting a process
  - How does a process know which others are involved in a deadlock ?
  - Can two process detect the same deadlock simultaneously ?
  - Use Priorities!
  - Rollback – release resources, clean up graph
- Phantom Deadlocks.