Distributed Deadlocks

Prof. Ananthanarayana V.S.
Dept. of Information Technology
N.I.T.K., Surathkal
Objectives of This Module

• In this module different kind of resources, different kind of resource request models are addressed

• Deadlock solutions – mainly prevention strategy and detection & recovery strategy in distributed environment is discussed
Roadmap

• Definitions
• Resource requests models
• Deadlock handling strategies
• Distributed deadlock handling algorithms
Definitions

• **Resources:**
  – There are two types of resources in computer system

• **Reusable resources**
  – They are fixed in number, neither can be created nor can be destroyed
  – To use the resource, the process must *request* for it, hold it during the usage (*allocation*) and *release* it on completion
  – The released resources may be re-allocated to other process
  – Example: Memory, CPU, Printer, Disk blocks ...etc

• **Consumable resources**
  – These resources will vanish once they are consumed
  – Producer process can produce any number of consumable resources if it is not blocked
  – Example: Messages, Interrupt signals, V operation in semaphore ...etc
Type of Resource Accesses

• **Shared**
  – In this mode, the resource can be accessed by any number of processes simultaneously
  – Example: Read lock on data item

• **Exclusive**
  – In this mode, the resource can be accessed by only one process at any point of time
  – Example: Write lock on data item
  – In theory of deadlocks, mostly exclusive locks are considered
  – Reusable resources can be accessed in exclusive or shared mode at a time
  – Consumable resources always accessed on exclusive mode
Resource Request Model

• Single unit resource request model
  – In this model, a process is allowed to request only one unit of the resource at a time. The process is blocked till that resource is allocated.
  – Example: A transaction (process) request for the write lock on data item [write_lock(X)]

• AND request model
  – In this model, a process is allowed to request multiple resources simultaneously. It is blocked till all the resources are available.
  – Example: Consider the data item X is replicated at N sites. A transaction request for write lock on X need to request for locks at all N where X is located and is blocked till all the write request is granted
Resource Request Model – Contd.

• OR request model
  – In this model, a process is allowed to request the multiple resources simultaneously. However, it is blocked till at-least one resource is allocated.
  – Example: Consider the data item X is replicated at N sites. A transaction request for read lock on X need to request for locks at all N where X is located. However, the transaction is blocked till at least one of the read lock request is granted.

• AND-OR request model
  – Here the request of the process is specified in the form of a predicate where its atoms / variables are the resources.
  – Example: R1 AND (R2 OR R3)

• P out of Q request model
  – Here, a process can simultaneously requests Q resources and will be blocked till any P out of Q resources are available.
  – Note that if P = 1, the model is OR request model; if P = Q, the model is AND request model.
A set of processes is said to be in deadlock state, if each of them is waiting for the resources to be released by the another process in the set.

**Necessary condition for the deadlock:**
- Mutual exclusion: - Non sharable characteristic of the resources. Ex: Memory location
- No pre-emption:- The allocated resources can’t be pre-empted from the process before its release by the process
- Hold and wait: The process holding some resources and waiting for other resources
- Circular wait:- The processes are waiting for one another for resources in a circular fashion

**Sufficient condition for the deadlock:**
- Note that the above mentioned conditions are not sufficient to say that the set of processes in deadlock. However, once the set of processes are in deadlock then we can observe all of those conditions. Hence they are necessary conditions.
Deadlock Handling Strategies

• Deadlock Prevention
  – Idea: Resources are granted to the requesting processes in such a way that there is no chance of deadlock (Vaccination). For example, allocating the resources requested if all available. Else wait for all. [So no hold and wait condition holds in that case.]

• Deadlock Avoidance
  – Idea: Resources are granted as and when requested by the processes provided the resulting system is safe. The system state is said to be safe, if there exist at least one sequence of execution for all the process such that all of them can run to completion without getting into deadlock situation.

• Deadlock Detection and Recovery
  – Idea: In this strategy, the resources are allocated to the processes as and when requested. However, the deadlock is detected by deadlock detection algorithm. If deadlock is detected, the system recovers from it by aborting one or more deadlocked processes.
Distributed Deadlock Algorithms
Distributed Deadlock Prevention

• Basic Idea:
  1. Each process is assigned a globally unique timestamp using Lamport’s logical clock, process number and site number [i.e., \(<\text{logical clock value, process id, site id}>\)].
  2. Every request from the process for the resource should accompany the process timestamp.
  3. The timestamp of requesting process (for the resource) is compared with the one who is holding the resource and suitable decision is made to prevent the occurrence of deadlock.
Algorithm for distributed deadlock prevention

- Suppose a resource R is held by P_1 at some site, and the process P_2 requests R. Let TS(P_1) and TS(P_2) are timestamps of P_1 and P_2 respectively.

- **Wait-die method:**
  - If TS(P_2) < TS(P_1) then P_2 waits /* P_2 is older */
  - Else P_2 is killed /* P_2 is younger */

**Distributed Deadlock Prevention**

POSSIBLE WAITING SEQUENCE FOR RESOURCES (Assumed that P_i waiting for some resource hold by P_{i-1})
Distributed Deadlock Prevention

— Note on Wait-die method:

1. P₂ waits if resource holder (i.e. P₁) is younger process
2. P₂ is killed if the resource holder is older process
3. Killed process will be restarted with SAME timestamp, will be older after some time and will not be killed
4. No circular wait condition will be hold in this method

   - The waiting sequence, TS(P₁) > TS(P₂) > ... > TS(Pₙ) leads to a circular wait provided P₁ waits for some resources hold by Pₙ, i.e., P₁ → Pₙ. This is possible only if TS(P₁) < TS(Pₙ). This contradict the waiting sequence, i.e., TS(P₁) > TS(Pₙ)
5. No preemption of process (resource holder) in this method. Here, the requester (P₂) will either waits or die.
Distributed Deadlock Prevention

– Wound-wait method:
  • If $TS(P_2) < TS(P_1)$ then $P_1$ is killed /* $P_2$ is older */
  • Else $P_2$ is waits /* $P_2$ is younger */

POSSIBLE WAITING SEQUENCE FOR RESOURCES (Assumed that $P_i$ waiting for some resource hold by $P_{i-1}$)

$TS(P_1) < TS(P_2) < \ldots < TS(P_n)$
Distributed Deadlock Prevention

– Note on Wound-wait method:

1. The older process never waits for the younger resource holder
2. $P_1$ is killed if the resource requester is older process
3. Killed process will be restarted with SAME timestamp, will be older after some time and will not be killed
4. No circular wait condition will be hold in this method
   - The waiting sequence, $TS(P_1) < TS(P_2) < ... < TS(P_n)$ leads to a circular wait provided $P_1$ waits for some resources hold by $P_n$, i.e., $P_1 \rightarrow P_n$. This is possible only if $TS(P_1) > TS(P_n)$. This contradict the waiting sequence, i.e., $TS(P_1) < TS(P_n)$
5. There is preemption of process (resource holder) in this method. Here, the requester ($P_2$) will wait or resource holder ($P_1$) will be wounded.
Distributed Deadlock Prevention

• Method to handle more than one process waiting on same resource (R):
  – Method 1:
    • At most only one process is allowed to wait for the resources and all other processes are killed. If another process \( P_3 \) requesting the same resource \( R \), then Wound-wait is applied between \( P_2 \) and \( P_3 \) to select oldest. Then, either Wound-wait or Wait-die method can be used between oldest waiting process and \( P_1 \), the resource holder.
  – Method 2:
    • The waiting processes are ordered in the increasing order of their timestamps. A new process requesting for the resource is made to wait if it is not the older than resource holder. If so, then Wound-wait method is applied between the new process and resource holder.
Distributed Deadlock Detection and Recovery

• Two components in this strategy:
  – Distributed Deadlock detection
  – Distributed Deadlock recovery

• Distributed Deadlock detection
  – Using Wait for graph (WFG)
    • WFG is a directed graph (V,E), where the vertices are the processes and the directed edge e_{ij} indicates that the process P_i is waiting for the resource hold by the process P_j.
    • The process P_i belong to any node in DCS.
    • All the resources are assumed to be single unit based
Distributed Deadlock Detection

- In Single Unit Resource Request Model:
  - A deadlock in this model is detected by existence of cycle in WFG
  - Note that a process can involve in only one cycle

![Diagram showing cycles and processes](image)
Distributed Deadlock Detection

• In AND Request Model:
  – A deadlock in this model is detected by existence of cycle in WFG
  – Note that a process can involve in more than one cycle

P1 \rightarrow P2 \rightarrow P3 \rightarrow P4
P1 \leftarrow P2 \leftarrow P3 \leftarrow P4
P3 in both cycles. Note that P3 is requested for the resources hold by P1 and P5. P3 is holding the resources requested by P2 and P6.
Distributed Deadlock Detection

• In OR Request Model:
  – A cycle in the WFG is not sufficient condition for the existence of the deadlock
  – Note that a process can involve in more than one cycle
In OR Request Model:

- A cycle in the WFG is not sufficient condition for the existence of the deadlock
- Note that a process can involve in more than one cycle

Cycles in WFG does not implies the deadlock situation. This is because, P₁ is requested for the resources hold by P₂, P₄ and P₅. Once it get the resource hold by P₄, the request edges from P₁ to P₄, P₁ to P₅ and P₁ to P₂ will be removed and hence there will not be any cycle and no deadlock.
Distributed Deadlock Detection

• OR Request Model
  – The necessary and sufficient condition for detecting the deadlock is the presence of knot.
  – Knot: A set of processes $S$ is said to be a knot, if
    • $\forall P_i \in S$,  
      – Dependency Set($P_i$) $\subseteq S$ and  
      – Dependency Set($P_i$) $\neq \emptyset$
  – Dependency Set of a process $P_i$ (DS($P_i$)): Set of all processes from which $P_i$ is expecting the unit of resources to be released.
  – Knot implies deadlock in any resource request model
Distributed Deadlock Detection

- OR Request Model: An Illustrative example

Here, $DS(P_1) = \{P_2, P_4, P_5\}$, $DS(P_2) = \{P_3\}$, $DS(P_3) = \{P_1\}$, $DS(P_4) = \{}$, $DS(P_5) = \{P_6\}$, $DS(P_6) = \{P_1\}$

Note that $S = \{P_1, P_2, P_3\}$ is not a knot, because $DS(P_1)$ is not in $S$. If you include $P_5$ and $P_4$ then, $S = \{P_1, P_2, P_3, P_4, P_5\}$ is again not a knot because $DS(P_4)$ is a null set. And so on.

Similar argument will follow for $S = \{P_1, P_5, P_6\}$ to show $S$ is not a knot.
Distributed Deadlock Detection

- In AND-OR Request Model:
  - Presence of KNOT in WFG implies that the system is in deadlock

- In P out of Q Request Model:
  - Presence of KNOT in WFG implies that the system is in deadlock
Requirements of Distributed Deadlock Detection Algorithm

• If there is a deadlock, then the algorithm should detect all such deadlocks. (i.e., algorithm should detect all deadlocks)

• If the algorithm says that there is a deadlock, then definitely there should one. (i.e., no false deadlock detection by the algorithm)
Pseudo Deadlock in Distributed Environment

- Let $P_1, P_2, ... P_n$ be the sequence of processes such that $P_i$ is waiting for the release of resources hold by $P_{i+1}$ (where $1 \leq i \leq n-1$).
- Let $P_n$ releases the resources first and then request for the resource hold by $P_1$. For this, $P_n$ will send the message (M1) to the resource controller to release the allocated resource for which $P_{n-1}$ is waiting and then send the message (M2) to the resource controller to allocate the resource hold by $P_1$.
- If M2 reaches first than M2 at deadlock detection algorithm, then there is a false / pseudo deadlock!
Distributed Deadlock Detection Algorithms

- Centralized Approach

- Distributed Approach: Chandy - Misra - Haas Algorithm
Centralized Approach

• The single unit request resource model is assumed in this approach. So, a cycle in the WFG implies the deadlock.
• The local wait for graph (LWFG) is constructed at each site.
• The global wait for graph (GWFG) is constructed at coordinator site based on following criteria:
  – Whenever an edge is added or deleted at LWFG, the local site will inform this to coordinator. Or
  – Periodically the local site will send its LWFG.
• The coordinator will construct the GWFG which is the concatenation of LWFGs.
• If there is a cycle, then the coordinator will detect that the system is in deadlock.
• But the problem is there is a possibilities of false deadlock as demonstrated in next slide.
Centralized Approach

At $S_1$:  $M_1 : T_3$ releases $R_2$

At $S_2$:  $M_2 : T_3$ requests  $R_3$

$M_1$ occurs before $M_2$

$M_2$ reaches coordinator before $M_1$
Centralized Approach

At $S_1$: $M_1 : T_3$ releases $R_2$
At $S_2$: $M_2 : T_3$ requests $R_3$

$M_1$ occurs before $M_2$
$M_2$ reaches coordinator before $M_1$

False Deadlock
Centralized Approach

At $S_1$: $M_1 : T_3$ releases $R_2$
At $S_2$: $M_2 : T_3$ requests $R_3$

$M_1$ occurs before $M_2$
$M_2$ reaches coordinator before $M_1$

False Deadlock
Centralized Approach

At $S_1$: \( M_1 : T_3 \) releases $R_2$
At $S_2$: \( M_2 : T_3 \) requests $R_3$
\{ $M_1$ occurs before $M_2$ $M_2$ reaches coordinator before $M_1$

**Solution:** Timestamp based messages; ordering at Coordinator
Handling of Pseudo Deadlock

• Pseudo deadlocks can be handled using timestamps based on Lamport’s clock value.
• Every messages pertaining to LWFG from the local site to the coordinator carries the timestamp.
• If the coordinator observe the cycle due to the message (M) from the local site, then the coordinator will broadcast that is there any message having the timestamp lesser than M?
• The decision about the cycle will be taken after the receipt of all the acknowledgements from the local site.
Handling of Pseudo Deadlock

• In the above example, since T₃ released R₂ and then requested R₃, M₁ timestamp should be smaller than M₂.

• When the coordinator receives M₂, it suspect the deadlock and it sends the message asking that if anyone has the message with timestamp lesser than M₂. For this, site S₁ will send the positive acknowledgement regarding M₁. The coordinator now reforms the GWFG with M₁ first and then M₂, hence no deadlock.
Chandy – Misra – Haas Algorithm

• Here, processes are allowed to request multiple resources at a time, so the process may wait for two or more resources.

• The process either wait for the resources hold by their co-processes (i.e., process in the same system) or by the processes exist in other machine.

• The algorithm is invoked when the process has to wait for the resource.

• For this, the process will generates probe message and send it to the process from whom it is waiting for the resource.
Chandy – Misra – Haas Algorithm

• The probe message consists of three components:
  – probe originator-id, sender-id, receiver-id
• When the process receives the probe message, it checks whether it is waiting for any process(s), If so, it update the probe message by updating its 2nd and 3rd fields and forward it to the process(es) for whom it is waiting.
• If the probe message goes all the way round and comes back to the originator (i.e., probe originator-id == receiver id), then the set of processes along the path of probe message are in deadlock.
• The probe initiator may identify itself as the victim and will commit suicide to break the deadlock.
Chandy – Misra – Haas Algorithm

The probe message initiated by process 0 is shown below: The arrow mark from process i to j indicates that process i is waiting for the resource hold by j.

Once process 0 receives the probe (0,8,0), then it realizes that it is in the set of processes under deadlock. So, it will identify itself to commit suicide to break the deadlock.
Chandy – Misra – Haas Algorithm

• The problem:
  – In the above example, it is possible that the processes, 0,1,2,3,4,6,8 may initiate the probe messages and identify themselves as the victim to commit suicide.
  – This leads to unnecessarily termination of many processes in the same deadlock path.

• Solution:
  – The process ids along the way is attached to the probe message in the form of a queue.
  – When the probe message comes back to the originator, it sees the process with highest / lowest process id in the queue and selects it as victim and sends the message to the victim to commit suicide by itself.
  – Even though many processes may initiate the probe messages, the same set of processes are identified in a cycle. Further, there is a single highest / lowest process id in that cycle. Hence, only one victim is selected.
Summary

• Generic resource request models are discussed

• Distributed deadlock prevention algorithms and distributed deadlock detection and recovery algorithms are outlined.
References

• Advanced Operating Systems

• Distributed Operating Systems
  – A S Tanenbaum, Prentice Hall

• Distributed Algorithms
  – Nancy A Lynch, Morgan Kaufman