Distributed Systems
13. Distributed Deadlock

Paul Krzyzanowski
Rutgers University
Fall 2017

Deadlock
Four conditions for deadlock
1. Mutual exclusion
2. Hold and wait
3. Non-preemption
4. Circular wait

Resource allocation
- Resource $R_i$ is allocated to process $P_i$
- $P_i$ holds $R_i$
- Resource $R_i$ is requested by process $P_i$
- $P_i$ wants $R_i$

This graph is called a Wait-For Graph (WFG)
Deadlock is present when the graph has cycles

Wait-For Graph: Deadlock Example
Circular dependency among four processes and four resources leads to deadlock

Dealing with deadlock
Same conditions for distributed systems as centralized
Harder to detect, avoid, prevent
Strategies
1. Ignore
   - Do nothing. So easy. So tempting.
2. Detect
   - Allow the deadlock to occur, detect it, and then deal with it by aborting and restarting a transaction that causes deadlock
3. Prevent
   - Make deadlock impossible by granting requests such that one of the conditions necessary for deadlock does not hold
4. Avoid
   - Choose resource allocation so deadlock does not occur (but algorithm needs to know what resources will be used and when)

Deadlock detection
- Kill off one or more processes when deadlock is detected
  - That breaks the circular dependency
- It might not feel good to kill a process
  - But transactions are designed to be abortable
- So just abort one or more transactions
  - System restored to state before transaction began
  - Transaction can restart at a later time
  - Resource allocation in the system may be different then so the transaction may succeed
Centralized deadlock detection

- Imitate the non-distributed algorithm through a coordinator
- Each system maintains a Wait-For Graph for its processes and resources
- A central coordinator maintains the combined graph for the entire system: the Global Wait-For Graph
  - A message is sent to the coordinator each time an edge (resource hold/request) is added or deleted
  - List of adds/deletes can be sent periodically

Centralized deadlock detection example

Two events occur:
1. Process $P_1$ releases resource $R$ on system $A$
2. Process $P_1$ asks system $B$ for resource $T$

Two messages are sent to the coordinator:
1 (from $A$): release $R$
2 (from $B$): wait for $T$

If message 2 arrives first, the coordinator constructs a graph that has a cycle and hence detects a deadlock. This is false deadlock.

Globally consistent (total) ordering must be imposed on all processes or Coordinator can reliably ask each process whether it has any release messages.

False Deadlock Example

No deadlock

Avoiding False Deadlock

Impose globally consistent (total) ordering on all processes or Have coordinator reliably ask each process whether it has any release messages

We detected deadlock because the coordinator received the messages out of order
Distributed deadlock detection

- Processes can request multiple resources at once
  - Consequence: process may wait on multiple resources
- Some processes wait for local resources
- Some processes wait for resources on other machines
- Algorithm invoked when a process has to wait for a resource

Distributed detection algorithm

Chandy-Misra-Haas algorithm

Edge Chasing
When requesting a resource, generate a probe message
- Send to all process(es) currently holding the needed resources
- Message contains three process IDs: (blocked ID, my ID, holder ID)
  1. Process that originated the message
  2. Process sending (or forwarding) the message
  3. Process to whom the message is being sent

Distributed deadlock detection
- When probe message arrives, recipient checks to see if it is waiting for any processes
  - If so, update & forward message: (blocked ID, my ID, holder ID)
  - Replace second field by its own process ID
  - Replace third field by the ID of the process it is waiting for
  - Send messages to each process on which it is blocked
- If a message goes all the way around and comes back to the original sender, a cycle exists
  - We have deadlock

Distributed deadlock prevention

Design system so that deadlocks are structurally impossible
- Deny circular wait
- Assign a unique timestamp to each transaction
- Ensure that the Global Wait-For Graph can only proceed from young to old or from old to young
### Deadlock prevention
- When a process is about to block waiting for a resource used by another
  - Check to see which has a larger timestamp (which is older)
- Allow the wait only if the waiting process has an older timestamp (is older) then the process waited for
- Following the resource allocation graph, we see that timestamps always have to increase, so cycles are impossible.
- Alternatively: allow processes to wait only if the waiting process has a higher (younger) timestamp than the process waiting for.

### Wait-die algorithm
- Old process wants resource held by a younger process
  - Old process waits
- Young process wants resource held by older process
  - Young process kills itself

```
Old process
TS=123

Young process
TS=311
```

### Wound-wait algorithm
- Instead of killing the transaction making the request, kill the resource owner
- Old process wants resource held by a younger process
  - Old process kills the younger process
- Young process wants resource held by older process
  - Young process waits

```
Old process
TS=123

Young process
TS=311
```

Only permit younger processes to wait on resources held by older processes.

---

### The end