Distributed Mutual Exclusion

Ch 10 [BenA 06]

Distributed System
Distributed Critical Section
Ricart-Agrawala
Token Passing Ricart-Agrawala
Token Passing Neilsen-Mizuno

Lesson 10

(Generic) Distributed System

- Nodes have processes
- Communication channels between nodes
  - Each node connected to every other node
  - Two-way channel
  - Reliable communication channels
    - Provided by network layer below
    - Messages are not lost
    - Messages processed concurrently with other computations (e.g., critical sections)
  - Nodes do not fail
- Requirements reduced later on
  - courses on distributed systems topics

Unrealistic assumptions?
Not really...

(Generic) Distributed System

- Processes (nodes) communicate with (asymmetric) messages
  - Message arrival order is not specified
  - Transmission times are arbitrary, but finite
  - Message (header) does not include send/receiver id
  - Receiver does not know who sent the message
    - Unless sender id is in the message itself

Distributed Processes

- Sender does not block
- Receiver blocks (suspended wait) until message of the proper type is received
- Atomicity problems in each node is not considered here
  - Solved with locking, semaphores, monitors, ...
- Message receiving and subsequent actions are considered to be atomic actions
  - Atomicity within each system considered solved

Distributed Critical Section Problem

- Processes within one node
  - Problem solved before
- Processes in different nodes
  - More complex
- State
  - Control pointer (CP, PC, program counter)
  - Local and shared variable values
  - Messages
    - Messages, that have been sent
    - Messages, that have been received
    - Messages, that are on the way
  - Arbitrary time, but finite!
Two Approaches for Crit. Section

- **A)** Ask everybody for permission to see, if it is my turn now
  - Lots of questions/answers
- **B)** I’ll wait until I get the token, then it is my turn
  - Pass the token to next one (which one?), or keep it?
  - Wait until I get the token
  - Token (turn) goes around all the time
- Moves only when needed?
- Both approaches have advantages/disadvantages
  - Who is “everybody”? How do I know them?
  - How do I know who has the token?
  - What if node/network breaks down?
  - What if token is lost?

Both approaches have advantages/disadvantages

- Who is “everybody”? How do I know them?
- How do I know who has the token?
- What if node/network breaks down?
- What if token is lost?

Do not worry now about the token getting lost ...

Ricart-Agrawala for Distributed Mutex

- Distributed Mutex, 1981 (Lamport, 1978)
- Modification of Bakery algorithm with ticket numbers
- Idea
  - Must know all other processes/nodes competing for CS
  - Choose own ticket number, “larger than previous”
  - Send it to everybody else
  - Wait until permission from everybody else
  - Exactly one will always get permission
  - All others will wait
  - Do your CS
  - Give CS permission to everybody who was waiting for you

Ricart-Agrawala Example

- 3 processes, each trying to enter CS concurrently
  - No status information needed on who had CS last

Ricart-Agrawala Example (contd)

- Receive process runs at each node
  - What if Aaron's receive completes 1st? Last? Becky's? not yet?

Ricart-Agrawala Example (contd)

- Becky executes CS, and then sends deferred replies to Aaron & Chloe
  - Aaron has now replies from everybody, and it can enter CS
  - What if Becky now selects ticket number 8, and requests CS?
    - Aaron's and Chloe's receive will both reply immediately? Ouch!

Algorithm 10.1: Ricart-Agrawala algorithm (outline)

```c
integer myNum = 0
set of node IDs, deferred = empty set

main application process, needs distributed mutex

p1: non-critical section
p2: for all other nodes N
p3: send(request, N, myID, myNum)
    await reply's from all other nodes
p4: critical section
p5: for all N in deferred
p6: receive N, unset deferred
    for all those waiting for my permission
p7: receive request, source, myNum
p8: if myNum < myNum
    send(reply, source, myID)
    take those wait by not sending reply
```

Ricart-Agrawala Example (contd)

- Becky executes CS, and then sends deferred replies to Aaron & Chloe
  - Aaron has now replies from everybody, and it can enter CS
  - What if Becky now selects ticket number 8, and requests CS?
    - Aaron's and Chloe's receive will both reply immediately? Ouch!

Ricart-Agrawala Example (contd)

- Becky executes CS, and then sends deferred replies to Aaron & Chloe
  - Aaron has now replies from everybody, and it can enter CS
  - What if Becky now selects ticket number 8, and requests CS?
    - Aaron's and Chloe's receive will both reply immediately? Ouch!

Ricart-Agrawala Example (contd)

- Becky executes CS, and then sends deferred replies to Aaron & Chloe
  - Aaron has now replies from everybody, and it can enter CS
  - What if Becky now selects ticket number 8, and requests CS?
    - Aaron's and Chloe's receive will both reply immediately? Ouch!
How to select ticket numbers

- Select always larger one than you have seen before
  - Larger than your previous myNum
  - Larger than any requestedNum that you have seen
- They all came before you, and you should not try to get ahead of them

What if equal ticket numbers?
- Fixed priority, based on node/process id numbers
- Used only with equal ticket numbers to avoid deadlock
  - Just like in Bakery algorithm

Quiescent Nodes
- Nodes that do not try to enter CS (but they could)
  - They are still listed in "all other nodes"
  - Problem with initial value of myNum
    - Initial value zero?
    - Initial value N > 0; tickets numbers eventually will reach it
      - Cure: receive checks for tickets numbers only if main wants CS

Token Based Algorithms
- Problems with permission based algorithms
  - Need permission from everybody (very many?)
    - At least everybody active
  - Inactive participants (those not wanting in CS) slow you down
    - Need reply from all of them!
    - Lots of synchronization even if only one tries to get into CS
      - Lots of communication (many messages)
- Token based algorithms
  - Have token, that is enough
    - No synchronization with everybody else needed
  - Get token, send token is simple
    - Communicate only with a few (fewer) nodes
    - Scalable?
    - Mutex is trivial, how about deadlock and starvation?

Algorithm 10.2: Ricart-Agrawala algorithm

```
Main
loop forever
  p1: non-critical section
  p2: requestCS = true
  p3: myNum = highestNum + 1
  p4: for all other nodes N
   p5: send(request, N, myID, myNum)
  p6: requestCS = false
  p7: critical section
  p8: for all nodes N in deferred
   p9: remove N from deferred
  p10: send(reply, N, myID)

Receive
integer source, requestedNum
loop forever
  p1: receive(request, source, requestedNum)
  p2: if requestCS or requestedNum < myNum
   p3: send(request, source, myID)
  p4: else add source to deferred
```
Ricart-Agrawala Token-Pass Ideas
- Send token to next one only when I know that someone wants it
- o/w keep token until needed
- Keep local requested array for best knowledge for the most recent CS request times
- Update this based on received CS request messages
- Keep local granted array, that has precise knowledge when each node actually was last granted CS
- Update it only when CS granted
- Pass it with token to next node
- Only this granted array (with token) is exactly correct!
- Other nodes have (slightly) old granted array

Algorithm 10.3: Ricart-Agrawala token-passing algorithm

```java
boolean haveToken = true in node 0, false in others
integer array[NODES] requested ← [0, ..., 0] ← local data in node
integer array[NODES] granted ← [0, ..., 0] ← distributed global data
integer myNum ← 0
boolean inCS ← false
if exists N such that requested[N] > granted[N]
    if no one else wants token, I will keep it
    for some such N
    send/token N, granted haveToken ← false
Receive: server process runs all the time
integer source, reqNum
loop forever
    receive(request, source, reqNum)
    requested[source] ← max([requested[source], reqNum])
    if haveToken and not inCS
        send Token
        give also most recent granted
        Ticket number for newest request
        for CS that I know of
```

Neilsen-Mizuno Token Based Algorithm
- Ricart-Agrawala: token carries queue of waiting processes
- Token can be very large, which may be problematic
- Neilsen-Mizuno: virtual tree structure within the nodes
  - Implements the queue
  - Algorithm utilizes virtual spanning tree of nodes
    - Spanning tree: all nodes linked as a tree, no cycles
    - Simple token indicates “turn” for critical section
    - Parent link points to the direction of last node in line for CS
    - Deferred link points to next node in line for CS

Neilsen-Mizuno Example
- Fully connected nodes
- Chloe is in CS
- No one waits for CS
Neilsen-Mizuno Example (contd)

Chloe has token, nobody waits for it
- Aaron requests CS
  - Sends msg=(req, Aaron, Aaron) on parent link
  - Removes himself from parent spanning tree
- Becky receives msg, and forwards the request “upward”
  - Sends msg=(req, Becky, Aaron) to Chloe
  - Moves to new parent spanning tree, points to Aaron
- Aaron is now last to request CS

Chloe receives msg (req, Becky, Aaron)
- Chloe in CS, sets deferred field to Aaron
  and sets parent field to Becky
- Chloe was (also) last in line for CS
  - When Chloe completes CS, she will pass token to Aaron
    - Token transferred directly to the next process in line for critical section (if any)
      - Just token is passed, no big array with it

Chloe still has CS, Evan wants CS
- Sends (req, Evan, Evan) to Danielle
- Danielle sends (req, Danielle, Evan) to Chloe
- Chloe sends (req, Chloe, Evan) to Becky
- Becky sends (req, Becky, Evan) to Aaron
- Aaron makes a deferred link to Evan

Chloe completes CS, passes token to Aaron
- Aaron completes CS, passes token to Evan
- Evan completes CS, keeps token

Algorithm 10.4: Neilsen-Mizuno token-passing algorithm

integer parent ← (initialized to form a tree)
integer deferred ← 0
boolean holding ← true in the root, false in others

Main
loop forever
  if non-critical section
    if not holding
      send(request, parent, myID, myID)
    parent ← 0
    mulit latest request for CS
    receive(token)
    holding ← false
  critical section
    if deferred ≠ 0
      someone wants the CS next
    send(token, deferred)
    deferred ← 0
  else holding ← true

Receive (runs concurrently with main, mutex problems solved...)
integer source, originator
loop forever
  if request, source, originator
  if parent = 0
    last in queue
  if holding
    have token, not in CS
  send(token, originator)
  holding ← false
  else deferred ← originator
  else send(request, parent, myID, originator)
  forward request
  parent ← source
  update direction for last request
  done

Ricart-Agrawala vs. Neilsen-Mizuno

- Number of messages needed?
- Size of messages?
- Size of data structures in each node?
- Behaviour with heavy load?
  - Many need CS at the same time
- Behaviour with light load?
  - Requests for CS do not come often
  - Usually only one process requests CS at a time

Other Distributed Mutex Algorithms

- Other token-based algorithms
  - Token ring: token moves all the time
  - Lots of token traffic even when no CS requests
- Centralized server
  - Simple, not very many messages
  - Not scalable, may become bottleneck
- Give up unrealistic assumptions
  - Nodes may fail
  - Messages may get lost, token may get lost
- See other courses

Summary

- Distributed critical section is hard, avoid it
  - Use centralized solutions if possible?
- Permission based solutions
  - Ricart-Agrawala – ask everyone
- Token based solutions
  - Ricart-Agrawala – centralized state in granted[]
  - Neilsen-Mizuno – queue kept in spanning tree
- There are other algorithms
- How do they scale up?