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

- Ask everybody for permission, if it is my turn now
  - Lots of questions/answers
- I’ll wait until I get the token, then it is my turn
  - Pass the token to next one (which one?)
  - 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?
  - What if someone does not talk to me?
  - What if node/network breaks down?
  - What if token is lost?

Two Approaches

- Ask everybody for permission, if it is my turn now
  - Lots of questions/answers
- I’ll wait until I get the token, then it is my turn
  - Pass the token to next one (which one?)
  - 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?
  - What if someone does not talk to me?
  - What if node/network breaks down?
  - What if token is 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 from everybody else?
  - All others will wait
  - Do your CS
  - Give CS permission to everybody else who was waiting for you

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 from everybody else?
  - All others will wait
  - Do your CS
  - Give CS permission to everybody else who was waiting for you

Algorithm 10.1: Ricart-Agrawala algorithm (outline)

```
main
    application process, needs distributed mutex
    integer myNum = 0
    set of node IDs defined = empty set
    for all other nodes N do not know
    send request, N, myID, myNum
    wait reply from all other nodes
    for all other nodes N do not know
    receive server process, runs concurrently all the time
    integer source, reqNum
    receive request, source, myNum
    if reqNum < myNum, send reply, source, myID
    else add source to deferred
    if I got reply from everybody, I can enter CS
    if I got reply from everybody, I can enter CS
    request run at each node
    what if Aaron’s request completes 1st? Last? Becky’s? not yet?
    if reqNum < myNum, send reply, source, myID
    else add source to deferred
    get myNum
    if myNum < most recent myNum, can enter CS
    deferred
    request run at each node
    what if Aaron’s request completes 1st? Last? Becky’s? not yet?
    if reqNum < myNum, send reply, source, myID
    else add source to deferred
```

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 from everybody else?
  - All others will wait
  - Do your CS
  - Give CS permission to everybody else 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

```
1. Receive process runs at each node
   what if Aaron’s receive completes 1st? Last? Becky’s? not yet?
2. if reqNum < myNum
   send(reply, source, myID)
   else add source to deferred
```

Ricart-Agrawala Example (contd)

- Receive process runs at each node
  - What if Aaron’s receive completes 1st? Last? Becky’s? not yet?
- if reqNum < myNum
  - Send(reply, source, myID)
  - Else add source to deferred

```
1. Receive process runs at each node
   what if Aaron’s receive completes 1st? Last? Becky’s? not yet?
2. if reqNum < myNum
   send(reply, source, myID)
   else add source to deferred
```
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

Discussion A

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

Discussion B

Token Based Algorithms

- Problems with permission based algorithms
  - Need permission from everybody (very many?)
  - 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 (faster) nodes
    - Scalable?
  - Mutex is trivial, how about deadlock and starvation?
Ricart-Agrawala ideas

- Send token to next one only when I know that someone wants it
  - or 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 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

Ricart-Agrawala: token carries queue of waiting processes
- Token can be very large, which may be problematic
- 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 in line for CS
  - Deferred link points to next in line for CS

Neilsen-Mizuno: virtual tree structure within the nodes implements the queue

Algorithm 10.3: Ricart-Agrawala token-passing algorithm

<table>
<thead>
<tr>
<th>Algorithm 10.3: Ricart-Agrawala token-passing algorithm (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main</strong> application process, needs distr mutex</td>
</tr>
<tr>
<td>loop forever non-critical section</td>
</tr>
<tr>
<td>if not haveToken</td>
</tr>
<tr>
<td>myNum = myNum + 1</td>
</tr>
<tr>
<td>for all other nodes N</td>
</tr>
<tr>
<td>receive(token, granted)</td>
</tr>
<tr>
<td>haveToken = true</td>
</tr>
<tr>
<td>inCS = true</td>
</tr>
<tr>
<td>granted[myID] = myNum</td>
</tr>
<tr>
<td>sendToken = true</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>if haveToken, no delays</td>
</tr>
<tr>
<td>Request token from everybody</td>
</tr>
<tr>
<td>Many messages?</td>
</tr>
<tr>
<td>Just one very large message?</td>
</tr>
<tr>
<td>Mutex?</td>
</tr>
<tr>
<td>No deadlock?</td>
</tr>
<tr>
<td>No starvation?</td>
</tr>
<tr>
<td>“some” in sendToken?</td>
</tr>
<tr>
<td>Scalable?</td>
</tr>
<tr>
<td>Overflows?</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>Update one field</td>
</tr>
<tr>
<td>Only if someone wants it! Send granted/also.</td>
</tr>
</tbody>
</table>

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

- 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

Neilsen-Mizuno Example (contd)

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

---

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1:</td>
<td>int parent = (initialized to form a tree)</td>
</tr>
<tr>
<td>p2:</td>
<td>int deferred = 0</td>
</tr>
<tr>
<td>p3:</td>
<td>boolean holding = have token, not in CS</td>
</tr>
<tr>
<td>p4:</td>
<td>if not holding</td>
</tr>
<tr>
<td>p5:</td>
<td>send(request, parent, myID, myID)</td>
</tr>
</tbody>
</table>
| p6:  | mark latest request for CS (runs concurrently with main, mutex problems solved…)
| p7:  | receive(token) |
| p8:  | if holding = false |
| p9:  | holding = true |
| p10: | critical section |
| p11: | if deferred ≠ 0 |
| p12: | someone wants the CS next |
| p13: | send(token, deferred) |
| p14: | deferred = 0 |
| p15: | else holding = true |

---
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?