Slides for Chapter 15: Coordination and Agreement

From Coulouris, Dollimore, Kindberg and Blair
Distributed Systems: Concepts and Design
Edition 5, © Addison-Wesley 2012
Overview of Chapter

- Introduction
- Distributed mutual exclusion
- Elections
- Coordination and agreement in group communication (skip)
- Consensus and related problems (skip)
Introduction

Covers two areas:

• Coordinating actions in a distributed system
• Distributed processes agreeing on a result value

• Assumes reliable communication channels for simplicity (failure is masked by a reliable communication protocol)
• Detecting that a process has failed can be reliable or unreliable
• Use timeouts
• Unreliable: replies *unsuspected* or *suspected*
• Reliable: replies *unsuspected* or *failed*
Figure 15.1
A network partition

Crashed router
Overview of Chapter

- Introduction
- Distributed mutual exclusion
- Elections
- Coordination and agreement in group communication
- Consensus and related problems
Distributed mutual exclusion

Known as *critical section* problem:
- Only one process can be in critical section – the process with the *token* that allows access to the resource
- Operations include the following:
  - `enter()` – requests access; can be granted or blocked
  - `resourceAccess()` – access resource in critical section
  - `exit()` – leave critical section – other processes may now enter

Conditions:
- ME1 (safety) – at most one process in critical section
- ME2 (liveness) – requests eventually succeed
- ME3 (ordering) – requests follow happened-before relationship
Distributed mutual exclusion

Various algorithms:
• Central server algorithm
• Ring-based algorithm
• Algorithm using multicast and logical clocks
• Voting algorithm
• Others
Figure 15.2
Server managing a mutual exclusion token for a set of processes

1. Request token
2. Release token
3. Grant token

Queue of requests

$p_1$

$p_2$

$p_3$

$p_4$
Figure 15.3
A ring of processes transferring a mutual exclusion token
Figure 15.4

Ricart and Agrawala’s algorithm

On initialization
    state := RELEASED;

To enter the section
    state := WANTED;
    Multicast request to all processes;
    \( T := \) request’s timestamp;
    Wait until (number of replies received = \( N - 1 \));
    state := HELD;

On receipt of a request \(<T_i, p_i>\) at \( p_j \) (\( i \neq j \))
    if (state = HELD or (state = WANTED and \( (T, p_j) < (T_i, p_i) \)))
       then
          queue request from \( p_i \) without replying;
       else
          reply immediately to \( p_i \);
    end if

To exit the critical section
    state := RELEASED;
    reply to any queued requests;
Figure 15.5
Multicast synchronization
On initialization
  state := RELEASED;
  voted := FALSE;

For \( p_i \) to enter the critical section
  state := WANTED;
  Multicast request to all processes in \( V_i \);
  Wait until (number of replies received = \( K \));
  state := HELD;

On receipt of a request from \( p_i \) at \( p_j \)
  if (state = HELD or voted = TRUE)
    queue request from \( p_i \) without replying;
  else
    send reply to \( p_i \);
    voted := TRUE;
  end if

For \( p_i \) to exit the critical section
  state := RELEASED;
  Multicast release to all processes in \( V_i \);

On receipt of a release from \( p_i \) at \( p_j \)
  if (queue of requests is non-empty)
    remove head of queue – from \( p_k \), say;
    send reply to \( p_k \);
    voted := TRUE;
  else
    voted := FALSE;
  end if
Overview of Chapter

• Introduction
• Distributed mutual exclusion
• Elections
• Coordination and agreement in group communication
• Consensus and related problems
Elections

Election algorithms:
• Used to choose a particular process for a role – for example, to choose a central server for distributed algorithms that require a central server
• Process can call an election; for example, if it detects that central server has failed

Requirements:
• E1: (safety) only one process is elected – each participant sets elected\(i\) either to P or to undefined if it does not know the elected process yet (P will be the participant with the largest process id)
• E2: (liveness) all processes participate and either set elected\(i\) to the elected process or crash
Elections

Election algorithms:
• Ring-based algorithm
• Bully algorithm
Figure 15.7
A ring-based election in progress

Note: The election was started by process 17.
The highest process identifier encountered so far is 24.
Participant processes are shown in a darker colour.
Figure 15.8
The bully algorithm

The election of coordinator $p_2$, after the failure of $p_4$ and then $p_3$
Overview of Chapter

• Introduction
• Distributed mutual exclusion
• Elections
• Coordination and agreement in group communication (skip)
• Consensus and related problems
Figure 15.9
Reliable multicast algorithm

On initialization

Received := {};

For process \( p \) to R-multicast message \( m \) to group \( g \)

\( B\)-multicast\((g, m)\); // \( p \in g \) is included as a destination

On \( B\)-deliver\((m)\) at process \( q \) with \( g = \text{group}(m) \)

if \( m \notin \text{Received} \)
then

Received := Received \( \cup \) \{ \( m \} \);

if \( q \neq p \) then \( B\)-multicast\((g, m)\); end if

\( R\)-deliver \( m \);

end if
Figure 15.10
The hold-back queue for arriving multicast messages
Notice the consistent ordering of totally ordered messages $T_1$ and $T_2$, the FIFO-related messages $F_1$ and $F_2$ and the causally related messages $C_1$ and $C_3$ – and the otherwise arbitrary delivery ordering of messages.
### Figure 15.12
Display from bulletin board program

<table>
<thead>
<tr>
<th>Item</th>
<th>From</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>A. Hanlon</td>
<td>Mach</td>
</tr>
<tr>
<td>24</td>
<td>G. Joseph</td>
<td>Microkernels</td>
</tr>
<tr>
<td>25</td>
<td>A. Hanlon</td>
<td>Re: Microkernels</td>
</tr>
<tr>
<td>26</td>
<td>T. L’Heureux</td>
<td>RPC performance</td>
</tr>
<tr>
<td>27</td>
<td>M. Walker</td>
<td>Re: Mach</td>
</tr>
</tbody>
</table>

end
Figure 15.13
Total ordering using a sequencer

1. Algorithm for group member $p$

On initialization: $r_g := 0$;

To TO-multicast message $m$ to group $g$
   B-multicast($g \cup \{\text{sequencer}(g)\}$, <$m$, $i$>);

On B-deliver(<$m$, $i$>) with $g = \text{group}(m)$
   Place <$m$, $i$> in hold-back queue;

On B-deliver($m_{\text{order}} =$ <$“order”, $i$, $S$>)) with $g = \text{group}(m_{\text{order}})$
   wait until <$m$, $i$> in hold-back queue and $S = r_g$;
   TO-deliver $m$;       // (after deleting it from the hold-back queue)
   $r_g = S + 1$;

2. Algorithm for sequencer of $g$

On initialization: $s_g := 0$;

On B-deliver(<$m$, $i$>) with $g = \text{group}(m)$
   B-multicast($g$, <$“order”, $i$, $s_g$>);
   $s_g := s_g + 1$;
Figure 15.14
The ISIS algorithm for total ordering
Figure 15.15
Causal ordering using vector timestamps

Algorithm for group member $p_i \ (i = 1, 2\ldots, N)$

**On initialization**

$V_i^g[j] := 0 \ (j = 1, 2\ldots, N);$ 

*To CO-multicast message $m$ to group $g*$

$V_i^g[i] := V_i^g[i] + 1;$

$B$-multicast($g$, \(<V_i^g, m>\));

*On B-deliver($<V_j^g, m>$) from $p_j$, with $g = \text{group}(m)$*

place $<V_j^g, m>$ in hold-back queue;

wait until $V_j^g[j] = V_i^g[j] + 1$ and $V_j^g[k] \leq V_i^g[k] \ (k \neq j);$ 

$CO$-deliver $m$; \hspace{1cm} // after removing it from the hold-back queue 

$V_i^g[j] := V_i^g[j] + 1;$
Figure 15.16
Consensus for three processes

Consensus algorithm

P_1

\[ v_1 = \text{proceed} \]

\[ d_1 := \text{proceed} \]

P_2

\[ v_2 = \text{proceed} \]

\[ d_2 := \text{proceed} \]

P_3 (crashes)

\[ v_3 = \text{abort} \]
Algorithm for process $p_i \in g$; algorithm proceeds in $f + 1$ rounds

*On initialization*

$$\text{Values}^1_i := \{v_i\}; \text{Values}^0_i = \{\}$$

*In round* $r$ ($1 \leq r \leq f + 1$)

- **B-multicast(g, Values}^r_i - \text{Values}^{r-1}_i\); // Send only values that have not been sent
- **Values}^{r+1}_i := \text{Values}_i^r\;**

*while (in round r)*

```
{  

    **On B-deliver(V_j) from some p_j**

    **Values}^{r+1}_i := Values}^{r+1}_i \cup V_j;** 

}
```

*After* $(f + 1)$ *rounds*

Assign $d_i = \text{minimum}(\text{Values}^{f+1}_i)$;
Faulty processes are shown coloured
Figure 15.19
Four Byzantine generals

Faulty processes are shown coloured