CS 455: INTRODUCTION TO DISTRIBUTED SYSTEMS
[DISTRIBUTED MUTUAL EXCLUSION]

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Yes. But what really is a second?
  - 1 second == time for a cesium 133 atom to make 9,192,631,770 transitions

- Even though we are assuming processes do not fail, how would you cope with tokens that are lost in transit?
Topics covered in this lecture

- Distributed Mutual Exclusion
  - Multicast & logical clocks [Agarwala & Ricart]
  - Maekawa’s voting based algorithm
- Election algorithms

Requirements for distributed mutual exclusion

- ME1: At most one process may execute in the critical section at a time
  - Safety
- ME2: Requests to enter and exit the critical section eventually succeed
  - Liveness: Freedom from deadlocks and starvation
- ME3: If one request happened-before another, then entry to the CS is granted in that order
Evaluation of the algorithms

- **Bandwidth consumed**
  - Proportional to number of messages sent in each entry and exit operation

- **Client delay** incurred by process for each entry or exit operation

- **Effect on throughput** of the system
  - **Synchronization delay** between one process exiting critical section and next process entering it
  - Throughput is greater when synchronization delay is shorter

---

**Mutual Exclusion using Multicast & Logical Clocks**  
\{Ricart & Agarwala's Algorithm\}
Agarwala & Ricart’s algorithm using multicast and logical clocks

- Processes that require entry to a critical section **multicast** a request message
  - *Enter it only when* all other processes have replied to request
- Process’ replies to a request are designed to ensure that ME1, ME2, and ME3 are met

The setting

- Processes $p_1, p_2, \ldots, p_N$ have distinct identifiers
- Processes have communication channels to each other
- Each process $p_i$ keeps a Lamport clock
- Messages requesting entry are of the form $<T, p_i>$
  - $T$ is the sender’s timestamp and $p_i$ is the sender’s identifier
Each process records its state

- Released
  - Outside the critical section
- Wanted
  - Wanting entry into the critical section
- Held
  - Being in the critical section

Entering the critical section

- If a process requests entry and the state of all other processes is Released
  - All processes respond immediately and the entry is granted
- If a process requests entry and some process is in the state Held
  - That holding process will not reply to requests until it has finished with the critical section
  - All other processes respond
Entering the critical section

- If two or more processes request entry at the same time?
  - Request with the lowest timestamp will be first to collect N-1 replies
  - If the Lamport timestamps are the same?
    - Requests are ordered based on their identifiers

- When a process requests entry?
  - Defers all processing requests from other processes until its own request has been sent

Multicast synchronization

Initial Condition:
- \( p_3 \) not interested in entering critical section
- \( p_1 \) and \( p_2 \) request entry concurrently
- Timestamp of \( p_1 \)'s request: 41
- Timestamp of \( p_2 \)'s request: 34

\( p_2 \) enters the critical section
Achieving the properties ME1, ME2 and ME3

- If two processes $p_i$ and $p_j$ ($i \neq j$) enter critical section at the same time?
  - Both these processes would have replied to each other; but the pairs $<T_i, p>$ are totally ordered
    - So it's impossible

- Requests to enter and exit the critical section eventually succeed because requests are served based on timestamps
  - Satisfies ME2 and ME3 (order)

Evaluation of the algorithm

- Gaining entry takes $2(N-1)$ messages
  - $N-1$ to multicast the request Followed by $N-1$ replies
  - Expensive in terms of bandwidth utilization

- Synchronization delay
  - Just one message transmission time
    - Previous algorithms incurred round-trip delays
Some observations [1/2]

- One of the problems with the central server algorithm was that it was a single point of failure.
- Here, the single point of failure has been replaced by N points of failure.
  - If any process crashes, it will fail to respond to requests.
    - This silence is interpreted (incorrectly) as a denial of permission.
    - Blocks ALL subsequent processes from entering the critical section.
- Solution: To have timeout mechanisms in place.

April 10, 2018
Instructor: SHRIDEEP PALICKARA
CS455: Introduction to Distributed Systems [Spring 2018]
Dept. Of Computer Science, Colorado State University

Some observations [2/2]

- Another problem with the central server algorithm was that making it handle all requests can lead to a bottleneck.
- In this setup all processes are involved in all decisions.
- Improvements?
  - Getting permission from everyone is an overkill.
  - All we need is to prevent two processes from entering the CS at the same time.

April 10, 2018
Instructor: SHRIDEEP PALICKARA
CS455: Introduction to Distributed Systems [Spring 2018]
Dept. Of Computer Science, Colorado State University
Maekawa’s solution to distributed mutual exclusion

- In order for a process to enter a critical section it is not necessary for all peers to grant access
  - Obtain permission from subsets of peers
  - Subsets used by any two peers must overlap

- Candidate process must collect sufficient votes to enter critical section
How mutual exclusion is achieved

- Processes at the intersection of two sets of voters ensure this
- Cast votes for only one candidate

Voting sets

- There is a voting set $V_i$ associated with each process $p_i$ (i = 1, 2, ..., N)

$$V_i \subseteq \{p_1, p_2, ..., p_N\}$$
Voting sets

- The sets \( V_i \) are chosen such that, for all \( i, j = 1, 2, \ldots, N \)
  
  \[
  p_i \in V_i \\
  V_i \cap V_j \neq \emptyset \\
  |V_i| = K
  \]

  To be fair, each process has a voting set of the same size

  Each process \( p_j \) is contained in \( M \) of the voting sets \( V_i \)

The optimal solution to the Maekawa’s algorithm

\[
K \sim \sqrt{N} \\
M = K
\]

Each process is in as many of the voting sets as there are elements in one of the sets
Calculation of voting sets

- Is not trivial
- As an approximation
  - Place processes in a $\sqrt{N}$ by $\sqrt{N}$ matrix
  - Voting set $V_i$ is the union of the row and column containing $p_i$
  - Voting set size is then $\sim 2\sqrt{N}$

Maekawa’s voting sets

Example

<table>
<thead>
<tr>
<th>$K$</th>
<th>$N$</th>
<th>$R_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>$R_1 = {1, 2, 3, 4}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_2 = {1, 5, 6, 7}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_3 = {1, 8, 9, 10}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_4 = {1, 11, 12, 13}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_5 = {2, 4, 6}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_6 = {2, 5, 7}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_7 = {2, 6, 9, 12}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_8 = {2, 7, 10, 13}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_9 = {2, 8, 12, 16}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_{10} = {2, 9, 13, 17}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_{11} = {3, 4, 7}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_{12} = {3, 5, 10, 12}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_{13} = {3, 6, 8, 13}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_{14} = {3, 7, 9, 11}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_{15} = {4, 5, 9, 13}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_{16} = {4, 6, 10, 11}$</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>$R_{17} = {4, 7, 8, 12}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$K$</th>
<th>$N$</th>
<th>$R_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>13</td>
<td>$R_1 = {1, 2, 3, 4, 5}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_2 = {1, 6, 7, 9, 9}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_3 = {1, 10, 11, 12, 13}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_4 = {1, 14, 15, 16, 17}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_5 = {1, 18, 19, 20, 21}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_6 = {2, 6, 10, 14, 18}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_7 = {2, 7, 11, 15, 19}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_8 = {2, 8, 12, 16, 20}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_9 = {2, 9, 13, 17, 21}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{10} = {3, 6, 11, 17, 20}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{11} = {3, 7, 10, 16, 21}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{12} = {3, 8, 13, 15, 18}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{13} = {3, 9, 12, 14, 19}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{14} = {3, 10, 15, 16, 20}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{15} = {4, 6, 12, 15, 21}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{16} = {4, 7, 13, 14, 20}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{17} = {4, 8, 10, 17, 19}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{18} = {4, 9, 11, 16, 18}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{19} = {4, 10, 13, 15, 17}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{20} = {4, 11, 14, 16, 19}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{21} = {4, 12, 17, 18, 20}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{22} = {4, 13, 16, 19, 21}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{23} = {5, 6, 13, 15, 17}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{24} = {5, 7, 12, 17, 19}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{25} = {5, 8, 11, 14, 21}$</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$R_{26} = {5, 9, 10, 15, 20}$</td>
</tr>
</tbody>
</table>

Entering the critical section

- To obtain entry into the critical section, each $p_i$ sends request message to all $K$ members of $V_i$
  - Including itself

- $p_i$ cannot enter critical section till it has received all $K$ reply messages

The reply message

- When a process $p_j$ in $V_i$ receives $p_i$'s request message it sends a reply message immediately unless ...
  - Its state is HELD
  - It has replied (voted) since it last received a release message
The release message

- To leave the critical section, $p_i$ sends **release message** to all $K$ members of $V_i$ (incl. itself)
- When a process receives a release message?
  - Removes the head of its queue of outstanding requests and sends a reply (vote) in response to it

Satisfying the safety property

- If it were possible for $p_i$ and $p_j$ to enter the critical section at the same time, then ...
  - Processes in $V_i \cap V_j \neq \emptyset$ would have voted for both $p_i$ and $p_j$
- But a process can make at **most one vote** between successive receipts of a release message
  - So it is impossible for $p_i$ and $p_j$ to both enter the critical section
But the basic algorithm is deadlock prone

- Consider three processes $p_1$, $p_2$, and $p_3$ with $V_1 = \{p_1, p_2\}$, $V_2 = \{p_2, p_3\}$, and $V_3 = \{p_3, p_1\}$

- If 3 processes concurrently request entry to the critical section it is possible for:
  - $p_1$ to reply to itself and hold-off $p_2$
  - $p_2$ to reply to itself and hold-off $p_3$
  - $p_3$ to reply to itself and hold-off $p_1$
  - Each process receives one of two replies; none can proceed

Resolving the deadlock issue

- Processes queue requests in the happened-before order
  - This also allows ME3 to be satisfied besides ME2
Analyzing the performance of the algorithm

- Bandwidth utilization
  - $2\sqrt{N}$ messages per entry into the critical section
  - $\sqrt{N}$ messages per exit
  - Total of $3\sqrt{N}$ is superior to $2(N-1)$ required by the previous algorithm (Ricart and Agarwala)
    - If $N \geq 3$

- Synchronization delay
  - Round-trip time

---

**ELECTION ALGORITHMS**
Election algorithms

- Algorithm for choosing a unique process to play a particular role
- When an elected process wants to retire, another election is needed

Calling an election

- When a process calls an election it initiates a particular run of the election algorithm
- A given process does not call more than one election at a time
  - With $N$ processes there could be $N$ concurrent elections
- At any point a process $p_i$ is either:
  - A participant: Engaged in the election algorithm
  - Non-participant: Not engaged in the election algorithm
The choice of the elected process must be unique

- Even in cases where several processes call the election simultaneously
- E.g., 2 processes see a coordinator has failed and they both call elections

The elected process is the one with the largest identifier

- The identifier is any value with the provision that the identifiers are unique and totally ordered
- E.g., electing process with the lowest computational load
  - Use $<\text{load}, i>$ as the identifier
  - Process $i$ is used to order identifiers with same load
Managing the identity of the elected process

- Each process $p_i$ ($i=1, 2, ..., N$) has a variable $\text{elected}_i$
  - Contains identifier of the elected process

- When a process first becomes a participant in an election
  - Set this variable to $\perp$ indicating that it is undefined

Requirements for the election algorithm

- **E1** (safety)
  - Participant process has $\text{elected}_i = \perp$ or $\text{elected}_i = P$
    - $P$ is a non-crashed process at the end of run with the largest identifier

- **E2** (liveness)
  - All processes $p_i$ participate and eventually either set $\text{elected}_i \neq \perp$ or crash
Measuring performance of election algorithms

- Network **bandwidth utilization**
  - How many messages are sent?

- **Turnaround time** for the algorithm
  - Number of message transmissions between the initiation and termination of a run

The contents of this slide set are based on the following references
