Outline for February 6, 2001

1. Greetings and felicitations!
   a. Friday times good, also Tuesday 3-4:30. Please send me your preferences!

2. Global state
   a. Show problem of slicing state when something is in transit
   b. Define local state; \(send(m_{ij}) \in LS_i\) iff time of \(send(m_{ij})\) < current time of \(LS_i\); similar for receive
   c. \(transit(LS_i, LS_j)\); inconsistent\((LS_i, LS_j)\); consistent state is one with inconsistent set empty for all pairs \(LS_i, LS_j\)
   d. Consistent global state: Chandy-Lamport

3. Termination detection
   a. Haung

4. Differences with non-distributed algorithms
   a. no shared memory, no common clock
   b. unpredictable message delays

5. Types of algorithms
   a. non-token algorithms: Lamport, Ricart and Agrawala, Maekawa
   b. token-based: Singhal, Raymond
   c. detection and recovery

6. System model
   a. states: requesting (entry section), executing (critical section), idle (remainder section), idle token (like idle, but you have the token)
   b. site: many others requesting entry, all are queued and served one at a time

7. Solution assumptions
   a. process names can be integers
   b. messages received in the order sent, in a finite amount of time, and correctly
   c. any process can communicate with any other process

8. Solution requirements
   a. mutual exclusion
   b. no deadlocks (progress)
   c. no starvation (bounded wait)
   d. fairness (requests executed in the order made, or in the order they arrive in system)
   e. fault tolerance (if a system fails, the algorithm can recover and continue to function)

9. Performance measures
   a. Performance under varying loads (low, high)
   b. Best, average, worst case performance

10. Terminology for non-token based protocols
    a. Request set \(R_i\) for a process \(p_i\); set of nodes from which \(p_i\) must obtain permission to enter critical section
    b. Requests ordered by timestamps: \((time, pid)\); the pid is used to disambiguate equal timestamps
    c. Request sets satisfy the following conditions:
        i. pairwise non-null intersection property: for all \(1 \leq i, j \leq n\) with \(i \neq j\), \(R_i \cap R_j \neq \emptyset\)
        ii. equal effort rule: for all \(1 \leq i, j \leq n\), \(|R_i| = K\)
        iii. equal responsibility rule: \(p_j\) is contained in \(D\) number of \(R_i\)
        iv. for all \(1 \leq i \leq n\), \(p_j \in R_i\)

11. Obvious solution: pick a single controlling site
    a. Advantages: 3 messages per request (site REQUEST, controller REPL Y, site RELEASES)
    b. Disadvantages: single point of failure, congestion, controller does all the work

12. Lamport’s
    a. Request set is all processes
    b. Performance: \(3(n-1)\) messages

13. Ricart and Agrawala’s
a. Request set is all processes
b. Performance: $2(n-1)$ messages
Chandy-Lamport Global State Recording Protocol

Introduction
The goal of this distributed algorithm is to capture a consistent global state. It assumes all communication channels are FIFO. It uses a distinguished message called a marker to start the algorithm.

Protocol

$P_i$ sends marker
1. $P_i$ records its local state $LS_i$
2. For each $C_{ij}$ on which $P_i$ has not already sent a marker, $P_i$ sends a marker before sending other messages.

$P_i$ receives marker from $P_j$
1. If $P_i$ has not recorded its state:
   a. Record the state of $C_{ji}$ as empty
   b. Send the marker as described above
2. If $P_i$ has recorded its state $LS_i$
   a. Record the state of $C_{ji}$ to be the sequence of messages received between the computation of $LS_i$ and the marker from $C_{ji}$. 
Example

Here, all processes are connected by communications channels $C_{ij}$. Messages being sent over the channels are represented by arrows between the processes.

Snapshot $s_1$:
- $P_1$ records $LS_1$, sends markers on $C_{12}$ and $C_{13}$
- $P_2$ receives marker from $P_1$ on $C_{12}$; it records its state $LS_2$, records state of $C_{12}$ as empty, and sends marker on $C_{21}$ and $C_{23}$
- $P_3$ receives marker from $P_1$ on $C_{13}$; it records its state $LS_3$, records state of $C_{13}$ as empty, and sends markers on $C_{31}$ and $C_{32}$.
- $P_1$ receives marker from $P_2$ on $C_{21}$; as $LS_1$ is recorded, it records the state of $C_{21}$ as empty.
- $P_1$ receives marker from $P_3$ on $C_{31}$; as $LS_1$ is recorded, it records the state of $C_{31}$ as empty.
- $P_2$ receives marker from $P_3$ on $C_{32}$; as $LS_2$ is recorded, it records the state of $C_{32}$ as empty.
- $P_3$ receives marker from $P_2$ on $C_{23}$; as $LS_3$ is recorded, it records the state of $C_{23}$ as empty.

Snapshot $s_2$: now messages are in transit on $C_{12}$ and $C_{21}$.
- $P_1$ records $LS_1$, sends markers on $C_{12}$ and $C_{13}$
- $P_2$ receives marker from $P_1$ on $C_{12}$ after the message from $P_1$ arrives; it records its state $LS_2$, records state of $C_{12}$ as empty, and sends marker on $C_{21}$ and $C_{23}$
- $P_3$ receives marker from $P_1$ on $C_{13}$; it records its state $LS_3$, records state of $C_{13}$ as empty, and sends markers on $C_{31}$ and $C_{32}$.
- $P_1$ receives marker from $P_2$ on $C_{21}$; as $LS_1$ is recorded, and a message has arrived since $LS_1$ was recorded, it records the state of $C_{21}$ as containing that message.
- $P_1$ receives marker from $P_3$ on $C_{31}$; as $LS_1$ is recorded, it records the state of $C_{31}$ as empty.
- $P_2$ receives marker from $P_3$ on $C_{32}$; as $LS_2$ is recorded, it records the state of $C_{32}$ as empty.
- $P_3$ receives marker from $P_2$ on $C_{23}$; as $LS_3$ is recorded, it records the state of $C_{23}$ as empty.
Huang’s Termination Detection Protocol

Introduction

The goal of this protocol is to detect when a distributed computation terminates.

Notation

- \( n \) processes
- \( P_i \) process; without loss of generality, let \( P_0 \) be the controlling agent
- \( W_i \) weight of process \( P_i \); initially, \( W_0 = 1 \) and for all other \( i \), \( W_i = 0 \).
- \( B(W) \) computation message with assigned weight \( W \)
- \( C(W) \) control message sent from process to controlling agent with assigned weight \( W \)

Protocol

- \( P_i \) sends a computation message to \( P_j \)
  1. Set \( W_i' \) and \( W_j \) to values such that \( W_i' + W_j = W_i \), \( W_i' > 0 \), \( W_j > 0 \). \((W_i' \) is the new weight of \( P_i \).)
  2. Send \( B(W_j) \) to \( P_j \)

- \( P_j \) receives a computation message \( B(W) \) from \( P_i \)
  1. \( W_j = W_j + W \)
  2. If \( P_j \) is idle, \( P_j \) becomes active

- \( P_i \) becomes idle:
  1. Send \( C(W_i) \) to \( P_0 \)
  2. \( W_i = 0 \)
  3. \( P_i \) becomes idle

- \( P_i \) receives a control message \( C(W) \):
  1. \( W_i = W_i + W \)
  2. If \( W_i = 1 \), the computation has completed.

Example

The picture shows a process \( P_0 \), designated the controlling agent, with \( W_0 = 1 \). It asks \( P_1 \), \( P_2 \), and \( P_3 \) to do some computation. It sets \( W_1 \) to 0.3, \( W_2 \) to 0.2, and \( W_3 \) to 0.5. \( P_2 \) in turn asks \( P_4 \) and \( P_5 \) to do some computations. It sets \( W_4 \) to 0.1 and \( W_5 \) to 0.1. When \( P_5 \) terminates, it sends \( C(W_3) = C(0.1) \) to \( P_0 \), which changes \( W_0 \) to \( 0 + 0.1 = 0.1 \).
When \( P_3 \) terminates, it sends \( C(W_3) = C(0.5) \) to \( P_0 \), which changes \( W_0 \) to \( 0.1 + 0.5 = 0.6 \).
When \( P_4 \) terminates, it sends \( C(W_4) = C(0.1) \) to \( P_0 \), which changes \( W_0 \) to \( 0.6 + 0.1 = 0.7 \).
When \( P_1 \) terminates, it sends \( C(W_1) = C(0.3) \) to \( P_0 \), which changes \( W_0 \) to \( 0.7 + 0.3 = 1 \).

\( P_0 \) thereafter concludes that the computation is finished.

Total number of messages passed: 9 (one to start each computation, one to return the weight to the controlling node).
Lamport’s Distributed Mutual Exclusion Protocol

Introduction

Lamport’s scheme uses distributed clocks. Every process notifies all others when it wants to enter the region of mutual exclusion. The process desiring to go in, enters when all others trying to get in made their request later.

Notation

- $n$ processes $p_1, \ldots, p_n$
- $t_j$ timestamp

Protocol

1. To enter the critical section, $p_i$ sends $\text{REQUEST}(t_i, i)$ to all sites and puts the request on its queue.
2. When $p_i$ receives a $\text{REQUEST}(t_j, j)$ message, it returns a $\text{REPLY}(t'_j, j)$ to $p_j$ and puts the request on its queue.
3. When $p_i$ receives a $\text{RELEASE}(t_j, j)$ message, it deletes $p_j$’s request from the queue.
4. $p_i$ enters the critical section when both of the following conditions hold:
   a. $p_i$ has received a message with timestamp larger than $(t_i, i)$ from each of the other sites
   b. $p_i$’s request is at the front of its request queue
5. When $p_i$ leaves the critical section, it removes its request from the top of the queue and sends $\text{RELEASE}(t_i, i)$ to all sites and puts the request on its queue.

Example

There are three processes, $p_1, p_2,$ and $p_3$. $p_1$ and $p_3$ seek mutually exclusive access to a shared resource.

<table>
<thead>
<tr>
<th>who</th>
<th>action</th>
<th>what</th>
<th>whom</th>
<th>$C1$</th>
<th>$C2$</th>
<th>$C3$</th>
<th>$Q1$</th>
<th>$Q2$</th>
<th>$Q3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>sends</td>
<td>Q(10,1)</td>
<td>all</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td>Q(10,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>sends</td>
<td>Q(4,3)</td>
<td>all</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>Q(4,3)</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>receives</td>
<td>Q(10,1)</td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td>Q(10,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>sends</td>
<td>P(10,2)</td>
<td></td>
<td></td>
<td>p1</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>receives</td>
<td>Q(4,3)</td>
<td></td>
<td></td>
<td>p3</td>
<td>11</td>
<td>Q(4,3)Q(10,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>sends</td>
<td>P(11,2)</td>
<td></td>
<td></td>
<td>p3</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>receives</td>
<td>Q(4,3)</td>
<td></td>
<td></td>
<td>p3</td>
<td>11</td>
<td>Q(4,3)Q(10,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>sends</td>
<td>P(11,1)</td>
<td></td>
<td></td>
<td>p3</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>receives</td>
<td>Q(10,1)</td>
<td></td>
<td></td>
<td>p1</td>
<td>10</td>
<td>Q(4,3)Q(10,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>sends</td>
<td>P(10,3)</td>
<td></td>
<td></td>
<td>p1</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>receives</td>
<td>P(10,3)</td>
<td></td>
<td></td>
<td>p3</td>
<td>12</td>
<td>Q(4,3)Q(10,1)P(10,3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>receives</td>
<td>P(11,1)</td>
<td></td>
<td></td>
<td>p1</td>
<td>12</td>
<td>Q(4,3)Q(10,1)P(11,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>receives</td>
<td>P(11,2)</td>
<td></td>
<td></td>
<td>p3</td>
<td>13</td>
<td>Q(4,3)Q(10,1)P(11,1)P(11,2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>enters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>receives</td>
<td>P(10,2)</td>
<td></td>
<td></td>
<td>p2</td>
<td>12</td>
<td>Q(4,3)Q(10,1)P(10,2)P(10,3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>leaves</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Q(10,1)</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>receives</td>
<td>R(13,3)</td>
<td></td>
<td></td>
<td>p1,p2</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>enters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Q(10,1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>leaves</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Q(10,1)</td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>sends</td>
<td>R(13,1)</td>
<td></td>
<td></td>
<td>p1,p2</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>receives</td>
<td>R(13,1)</td>
<td></td>
<td></td>
<td>p1</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>receives</td>
<td>R(13,1)</td>
<td></td>
<td></td>
<td>p1</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ricart and Agrawala’s Distributed Mutual Exclusion Protocol

Introduction
Ricart and Agrawala’s protocol is an optimization of Lamport’s. They piggyback the release message onto the reply.

Notation
- \( n \) processes \( p_1, \ldots, p_n \)
- \( t_j \) timestamp

Protocol
1. To enter the critical section, \( p_i \) sends REQUEST\((t_i, i)\) to all sites.
2. When \( p_i \) receives a REQUEST\((t_j, j)\) message:
   a. if it is not trying to enter the region of mutual exclusion, it returns REPL\(Y\)(\(t'_j, j\)) to \( p_j \).
   b. if it is trying to enter the region of mutual exclusion, and if \((t_i, i) \Rightarrow (t_j, j)\), it retains the REQUEST.
   c. otherwise it returns a REPL\(Y\)(\(t'_j, j\)) to \( p_j \).
3. When \( p_i \) has received a REPL\(Y\) message from every other process, it enters the region of mutual exclusion.
4. When \( p_i \) leaves the region of mutual exclusion, it sends REPL\(Y\)(\(t_i, i\)) to all processes with deferred requests.

Example
There are three processes, \( p_1, p_2, \) and \( p_3. \) \( p_1 \) and \( p_3 \) seek mutually exclusive access to a shared resource.

<table>
<thead>
<tr>
<th>who</th>
<th>action</th>
<th>what</th>
<th>whom</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>sends</td>
<td>Q(10,1)</td>
<td>all</td>
<td>10</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>sends</td>
<td>Q(4,3)</td>
<td>all</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>receives</td>
<td>Q(10,1)</td>
<td>p1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>sends</td>
<td>P(10,2)</td>
<td>p1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>receives</td>
<td>Q(4,3)</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>sends</td>
<td>P(11,2)</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>receives</td>
<td>Q(4,3)</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>sends</td>
<td>P(11,1)</td>
<td>p3</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>receives</td>
<td>Q(10,1)</td>
<td>p1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>receives</td>
<td>P(11,1)</td>
<td>p1</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>receives</td>
<td>P(11,2)</td>
<td>p2</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>enters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>receives</td>
<td>P(10,2)</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td>Q(10,1)P(10,2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>leaves</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Q(10,1)</td>
</tr>
<tr>
<td>p3</td>
<td>sends</td>
<td>P(12,3)</td>
<td>p1</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>receives</td>
<td>P(12,3)</td>
<td>p3</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>enters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>leaves</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Last modified at 7:04 pm on Tuesday, February 27, 2001