Process Groups and Virtual Synchrony

EECS 591 - Handout
Reading List

• Virtual Synchrony & ISIS protocol: Tanenbaum 7.4.3
• “Totem: A Fault-Tolerant Multicast Group Communication System” by Moser et. al.
• “Understanding Limitations of CATOCS” by Cheriton & Skeen
• “A Response to Cheriton & Skeen’s Criticism of CATOCS” by Birman
Process Groups

- Process Groups: a paradigm for building fault-tolerant distributed systems based on two primitives –
  - Group Multicast Communication
  - Group Membership

- Group view refers to the set of processes in the group
- Processes join and leave a group --- view change
- Membership changes and multicast messages are interleaved

- **Virtual synchrony**: a strong execution model for process groups in which the same set of multicast messages are delivered between view changes events by all non-faulty members. Also membership (view) changes seen by all processes in the same order.
- What happens if there is a process failure in the middle of multicasting a message?
Virtual Synchrony (1)

The logical organization of a distributed system to distinguish between message receipt and message delivery.
Virtual Synchrony (2)

The principle of virtual synchronous multicast.

The diagram illustrates a scenario where P1 joins the group, and messages are exchanged among P1, P2, P3, and P4. P3 crashes, and then rejoins. The group sets G change accordingly. Partial multicast from P3 is discarded.
### Message Ordering (1)

<table>
<thead>
<tr>
<th>Process P1</th>
<th>Process P2</th>
<th>Process P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>sends m1</td>
<td>receives m1</td>
<td>receives m2</td>
</tr>
<tr>
<td>sends m2</td>
<td>receives m2</td>
<td>receives m1</td>
</tr>
</tbody>
</table>

Three communicating processes in the same group. The ordering of events per process is shown along the vertical axis.
### Message Ordering (2)

<table>
<thead>
<tr>
<th>Process P1</th>
<th>Process P2</th>
<th>Process P3</th>
<th>Process P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>sends m1</td>
<td>receives m1</td>
<td>receives m3</td>
<td>sends m3</td>
</tr>
<tr>
<td>sends m2</td>
<td>receives m3</td>
<td>receives m1</td>
<td>sends m4</td>
</tr>
<tr>
<td></td>
<td>receives m2</td>
<td>receives m2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>receives m4</td>
<td>receives m4</td>
<td></td>
</tr>
</tbody>
</table>

Four processes in the same group with two different senders, and a possible delivery order of messages under FIFO-ordered multicasting.
## Virtual Synchrony Message Orderings

<table>
<thead>
<tr>
<th>Multicast</th>
<th>Basic Message Ordering</th>
<th>Total-ordered Delivery?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable multicast</td>
<td>None</td>
<td>No</td>
</tr>
<tr>
<td>FIFO multicast</td>
<td>FIFO-ordered delivery</td>
<td>No</td>
</tr>
<tr>
<td>Causal multicast</td>
<td>Causal-ordered delivery</td>
<td>No</td>
</tr>
<tr>
<td>Atomic multicast</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>FIFO atomic multicast</td>
<td>FIFO-ordered delivery</td>
<td>Yes</td>
</tr>
<tr>
<td>Causal atomic multicast</td>
<td>Causal-ordered delivery</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Six different versions of virtually synchronous reliable multicasting.
Implementing Virtual Synchrony --- ISIS System

• ISIS System --- first group communication system built upon virtual synchronous execution models
• Group multicast communication and group membership services built on top of TCP
• Multicast semantics supported:
  • abcast: view-synchronous totally-ordered group multicast
  • cbcast: view-synchronous causally-ordered group multicast
  • cabcast: view-synchronous causally and totally-ordered group multicast
Implementing Virtual Synchrony --- ISIS System

- Main issue: ensure that all messages sent to view G are delivered to all non-faulty processes in G before next group view G is installed.
- If the sender fails before its message m is delivered to all members, processes that haven’t received the message must get it from other processes.
- How? Every process in G keeps m until it knows that all members in G have received it, i.e., m is stable.
- Upon receiving a view change message, a process forwards all of its unstable messages and then send a flush message for the new view.
- After a process receives a flush message for the new view from each other process, it can safely install the new view.
Implementing Virtual Synchrony

a) Process 4 notices that process 7 has crashed, sends a view change

b) Process 6 sends out all its unstable messages, followed by a flush message

c) Process 6 installs the new view when it has received a flush message from everyone else
What if?

• Another process fails during a view change?