Process Groups and Virtual Synchrony

EECS 591 - Handout

Reading List

- Failure Models: Tanenbaum 7.1.1-7.1.2
- Group Communication Tanenbaum 7.4
  - Reliable Multicast Group Communication 7.4.1
  - Scalability Issue 7.4.2
  - Virtual Synchrony & ISIS protocol 7.4.3
Failure Models

<table>
<thead>
<tr>
<th>Type of failure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash failure</td>
<td>A server halts, but is working correctly until it halts</td>
</tr>
<tr>
<td>Omission failure</td>
<td></td>
</tr>
<tr>
<td>Receive omission</td>
<td>A server fails to respond to incoming requests</td>
</tr>
<tr>
<td>Send omission</td>
<td>A server fails to receive incoming messages</td>
</tr>
<tr>
<td>Timing failure</td>
<td>A server's response lies outside the specified time interval</td>
</tr>
<tr>
<td>Response failure</td>
<td></td>
</tr>
<tr>
<td>Value failure</td>
<td>The server's response is incorrect</td>
</tr>
<tr>
<td>State transition failure</td>
<td>The value of the response is wrong</td>
</tr>
<tr>
<td>Arbitrary failure</td>
<td>The server deviates from the correct flow of control</td>
</tr>
<tr>
<td></td>
<td>A server may produce arbitrary responses at arbitrary times</td>
</tr>
</tbody>
</table>

Different types of failures.

An Archetypal Multicast API

Primitive operations:

- **Join** – This operation allows a process to join a specific multicast group. A process that has joined a multicast group is a member of the group and is entitled to receive all multicast addressed to the group. A process should be able to be a member of multiple multicast groups at any one time.

Note that for this and other multicast operations, a naming scheme is needed to uniquely identify a multicast group.
Multicast API Operations - continued

• **Leave** – This operation allows a process to stop participating in a multicast group. A process that has left a multicast group is no longer a member of the group and is thereafter not entitled to receive any multicast addressed to the group, although the process may remain a member of other multicast groups.

• **Send** – This operation allows a process to send a message to all processes currently participating in a multicast group.

• **Receive** – This operation allows a member process to receive messages sent to a multicast group.

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?
Basic Reliable-Multicasting Schemes

![Diagram of Reliable Multicasting]

A simple solution to reliable multicasting when all receivers are known and are assumed not to fail
a) Message transmission
b) Reporting feedback

Nonhierarchical Feedback Control

![Diagram of Nonhierarchic Feedback]

Several receivers have scheduled a request for retransmission, but the first retransmission request leads to the suppression of others.
Hierarchical Feedback Control

The essence of hierarchical reliable multicasting.

a) Each local coordinator forwards the message to its children.
b) A local coordinator handles retransmission requests.

Virtual Synchrony (1)

• The logical organization of a distributed system to distinguish between message receipt and message delivery
• Distinguish between message reception (by host) and delivery (to application)
Virtual Synchrony (2)

The principle of virtual synchronous multicast.

Why Virtual Synchrony?

- Replication
- Fault-tolerance
  - Protect against Byzantine failures
- Distributed Databases
  - Banking
  - Distributed Dynamic Content Servers
- Distributed Control Systems
  - Aircraft/Battleship controls
Reliable, Unordered Delivery

Three communicating processes in the same group.
The ordering of events per process is shown along the vertical axis.

<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>

FIFO Delivery

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

<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>receives m2</td>
<td>receives m2</td>
<td>receives m2</td>
<td></td>
</tr>
<tr>
<td>receives m4</td>
<td>receives m4</td>
<td></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.
Causal Delivery

- “Related” messages are delivered in order
- How you would like to receive email (and read Usenet)
- Implemented using vector clocks

<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 m2</td>
<td>sends m2</td>
</tr>
<tr>
<td>receives m2</td>
<td>receives m2</td>
<td>receives m1</td>
<td>sends m4</td>
</tr>
<tr>
<td>sends m3</td>
<td>receives m3</td>
<td>receives m3</td>
<td>receives m1</td>
</tr>
<tr>
<td>receives m4</td>
<td>receives m4</td>
<td>receives m4</td>
<td>receives m3</td>
</tr>
</tbody>
</table>

Total Ordered (Atomic) Delivery

- All processes see same delivery order
- Often used in replication

<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 m1</td>
<td>sends m3</td>
</tr>
<tr>
<td>sends m2</td>
<td>receives m3</td>
<td>receives m3</td>
<td>sends m4</td>
</tr>
<tr>
<td>receives m2</td>
<td>receives m2</td>
<td>receives m2</td>
<td></td>
</tr>
<tr>
<td>receives m4</td>
<td>receives m4</td>
<td>receives m4</td>
<td></td>
</tr>
</tbody>
</table>
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
  - **cabast**: 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?
  • Protocol fails – no flush message received from failed process

• How to fix?
  • Announce new view changes while old ones are pending
  • After messages are stable in most recently announced view change, install new view

How to Implement?

• Central Sequencer
  – Messages are assigned a sequence by a single authority
  – Easy to ensure consistency
  – Single point of failure
  – Limited Scalability

• Train (Token ring)
  – A special message (token) passed in well-known order
  – Processes can only send messages when they have token
  – No single point of failure
  – Scalability still limited -- latency
Improving Scalability

• Difficult to scale message ordering and group membership management
• Message latency often proportional to group size
• Failure overhead increases drastically

Improving Scalability: “Gossip”

• By Ken Birman (creator of ISIS)
• All processes can provide retransmissions
• When a message is lost, one process picked at random to retransmit (can be based on distance)
• Probabilistic reliability and performance
• Highly scalable
• Weak ordering and membership guarantees
Improving Scalability: Multiple Groups

- Processes divided into multiple process groups
- Inter-group communication maintains end-to-end connectivity
- Message ordering and reliability guarantees on inter-group communication
- Takes advantage of natural communication locality
- Less Scalable than Gossip
- Still provides strong ordering and membership guarantees