Distributed Systems
Reliable Group Communication

Group F

March 2013
Overview

The Basic Scheme

The Basic Scheme
Example

Feedback Control

Non-Hierarchical
Example
Hierarchical
Example

Atomic multicast

Virtual Synchrony
Example

Message Ordering

Implementing virtual synchrony
Example of a view change
The basic scheme

- Point-to-point connections.
- Assume these points:
  - We know if we are member of a group or not.
  - Processes do not fail.
  - Processes do not leave or join whilst communication is going on.
- This way, reliable multicasting is simply that each message should be delivered to all the members in the group.
Example

Figure 8-9. A simple solution to reliable multicasting when all receivers are known and are assumed not to fail. (a) Message transmission. (b) Reporting feedback.
Non-Hierarchical Feedback Control

The general idea of a non-hierarchical scheme is to reduce the amount of feedback messages received.

- We only report when we miss a message.
- We multicast that report to the rest of the group.
- This way, let all other members suppress their acknowledgement.
- Sender then multicasts message again.

Problems:

- Accurate scheduling, timing messages
- Processing useless messages, as members who have received messages are interrupted.
Example

Figure 8.10. Several receivers have scheduled a request for retransmission, but the first retransmission request leads to the suppression of others.
Hierarchical Feedback Control

To achieve scalability, we will need to have a solution which supports larger networks.

- Introducing the Coordinator, once again.
- Subgroups - communicating through the coordinator.
- Tree-structure

Problems:

- Construction of a tree dynamically.
Example

Figure 8-11. The essence of hierarchical reliable multicasting. Each local coordinator forwards the message to its children and later handles retransmission requests.
Atomic multicast

In a system where atomic multicast is present,

- a message is either delivered to all or none at all, and
- all messages are delivered in the same order to all processes.
Virtual Synchrony

In a system where virtual synchrony is implemented, either all get the message, or none. So, if a process sends a message $m$, then crashes, it’s ensured that either

- $m$ is delivered to the remaining processes, or
- $m$ is ignored by all processes (as if it hasn’t been sent).

A multicast can be seen as an epoch, separated by view / group membership changes (leaving or joining). Each message has a **group view** (The set $G$) associated, i.e. a list of all processes that need to receive the message.
Example of ignored message

P3 crashes before completing the multicast, P2 and P4 are ignoring the message from P3, treating P3 as if he crashed before sending anything.
Message Ordering

In general there are four different orderings:

1. Unorded multicast
   ▶ Nothing is guaranteed

2. FIFO-ordered multicast
   ▶ Ordering per process is guaranteed

3. Causally-ordered multicast
   ▶ Potential causality is preserved

4. Totally-ordered multicast
   ▶ They are delivered in the same order to all group members
Implementing Virtual Synchrony

**Isis**, a fault tolerant distributed system. Using TCP to ensure delivery. But TCP can’t ensure delivery to all in case of a crash. This is done by the communication layer, making sure that a message is **stable** (i.e. everyone has the message) before it’s passed on to the application.

To make sure that we have a stable system, an arbitrary (operational) process in G are asked to send out the message (in the case the sender has crashed)
A view change

Process 4 notices a view change and sends out a notice to everyone
A view change

Every process (here from the view of 6) sends every unstable message it currently has.
It then sends a flush message indicating that it’s clean, and its view is ready to be updated.
A view change

Once the process has received a flush message from all, it can update its view.