# Distributed algorithms for fault-tolerance

Groups and virtual synchrony

Simin Nadjm-Tehrani



## Last lecture

- How to deal with failure detection for consensus?
- Another approach: push it down!



## Why groups?

- To deal with networks that vary in time
- All networks will have failing nodes, need to model repairs too!
- Even if nodes are not replicas some common state may be needed (e.g. distributed search)

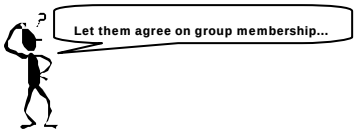


## Groups

- A general notion used to model
  - dynamic applications
  - recovery from failures (by rejoining)
- Membership service a main requirement

## ”Chicken and egg” again

- Implement membership service by a central server
- What if the server crashes?
- Replicate the server that serves … that serves…






## The ISIS system

- Only crash failures (no partitions)
- Distinguish between failed and slow processes:
  - special transport layer combining reliable delivery with failure detection
- A slow process is ”forced to fail” by exclusion from the *group*
- No recovery



## View updates

- P = $\{p_1, \ldots, p_n\}$
- processes are organised into sets of groups: G = $\{g_1, \ldots, g_m\}$
- view(g) recursively defined as a sequence:
  - $view_0(g) = \text{Æ}$
  - $view_i(g)$ Í P
  - $view_i(g)$ differs from $view_{i+1}(g)$ by addition/removal of one process



## Event ordering

- Assume a causal ordering ®

$p e ®p e´ Þ e ® e´

" m send(m) ® rec(m)

The communication layer delivers according to this order, synchronising at view updates

## Group-based delivery

send(m) ® send(m´) Þ
"p Î dest(m) Ç dest(m´)
deliver(m) ®p deliver(m´)



## ISIS implementations

- Check causal order algorithms in Schiper 1.6.2 and compare with section 5.1 in:
  [Birman, Schiper, Stephenson 91]
- Causal order delivery twice as fast as Atomic!
- Packet size dominant in determining cost



## What about?

- Synchronisation for changing group views?
- Does the replication style matter?



## Active replication

- A joining process needs to get the current state - use a state transfer mechanism
- Messages received prior to state update are buffered
- Replica computations need to be deterministic modulo causal order (atomic order) of delivery

## Primary-backup

- Need to identify the current primary in the group
- Need to update the backups, both for state and for group view
- On failure of primary, all or none!

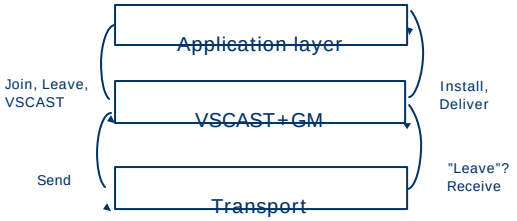


## Virtual synchrony

- The system treats one distributed event at a time
  - multicasts
  - group membership changes
  - failures



## View synchronous broadcast






## Group change atomicity

- Let $p_j \in g$ execute VSCAST(m,g) while in $view_i(g)$
- Either m is delivered in $view_i(g)$
- or a new $view_{i+1}(g)$ is installed
  - if p delivers m before installing $view_{i+1}(g)$, then
  - "$q \in view_{i+1}(g)$, q has delivered m before installing $view_{i+1}(g)$

## Flush algorithm

- After a view update message is received every member in the new group sends a flush message to all other members
- No new multicasts are started until earlier messages delivered and the new view is installed
- Check section 5.5 in Birman, Schiper and Stephenson, 1991



## Remarks

- What if Flush messages are not received?
- When do we know that earlier messages have been delivered?
- Is algorithm safe and live?



## Reading material

- Birman's CACM 1993 article
- Schiper chapter 1.6.1
- Birman, Schiper, Stephenson 1991