CS514: Intermediate Course in Computer Systems

Tracking Group Membership

We’ve seen . . .

- The concept of logical time
- How it can be used to build ordering into group communications systems
  - FIFO, Causal, Total (Agreed)
- Different forms of message reliability
  - Best effort, reliable, safe
- All in the context of a “static” group of processes
Today we’ll see …

- How to provide the processes in a process group with the group membership
  - As processes join and leave the group, and fail
- Using some of the tools we’ve already learned

Recall Virtual Synchrony: A series of “views”
Properties of views

- To be useful, a series of views requires certain properties:
  - Call $V$ the old view, and $V'$ the subsequent new view
  - At least one process in $V'$ must also have been in $V$
    - Obviously: otherwise the system state cannot be maintained across views

- The delivery semantics of messages sent but not delivered in view $V$ must be maintained in the new view $V'$ for all processes both in $V$ and $V'$.
  - Even when the sending process is not in $V'$
  - This allows the application to not worry about synchronizing system state in continuing processes
    - Only processes joining in view $V'$ need to be synchronized
Delivery semantics across views

Messages sent in view V must be completed before messages are sent in V’. This allows correct synchronization of joining processes. It also allows continuing processes to synchronize process failures with system state.
Message completion

ISIS message completion

ISIS:
NewView
Flush
Spread message completion

Properties of views: Two types of partitions

- A process group may partition
  - Processes in each partition are still alive, but cannot communicate with processes in other partitions
Properties of views: Two types of partitions

- Primary component model: One group is considered “primary”, and continues operation
  - All other groups “do nothing”, and try to rejoin the primary group (possibly as new processes)

- Simultaneous components model: All groups continue operation, later groups may merge

Primary component model

```
\[ V \]
\[ V' \]
\[ V'' \]
\[ P_1 \quad P_2 \quad P_3 \quad P_w \quad P_z \]
```
Simultaneous components model

In the case of ISIS, the primary component is the one with a majority of processes from the previous view. Note that no group may have a majority! In this case, the system is essentially restarted.

Typically a “partition” occurs when a single computer loses its network interface.

In LAN environment, it is not hard to prevent “non-trivial” (single node) partitions.

Primary component model
Simultaneous component model

- In many (most?) high availability environments, simultaneous components doesn’t make sense
  - Two partitioned groups cannot think they are controlling the same air space!
- Merging system state is hard
  - May have consistency issues!

Word of caution: some things are impossible!

- In general, group communications systems cannot tolerate all possible failures
  - Either theoretically or in practice
    - Non-majority partitions
    - Byzantine failures
    - Some “unfortunately timed” failures
- But in practice we can come close (and still get decent performance)
One-tier or two-tier group membership service

- One-tier: all processes in group participate in membership protocol
- Two-tier: a small group of processes offer a “group membership service”
  - Multiple “client” process groups subscribe to the service
  - The group member service keeps track of client group membership
  - The group member service also keeps track of its own membership

Two-tier group membership service
Why two tiers?

- Group membership protocol doesn't scale well
  - \( O(N^2) \)
  - Two- or even Three-phase commit
- Some groups may be large
- Some processes may be in multiple groups
- Therefore better to have one small dedicated group membership service

Two-tier GMS details

- GMS processes well-known (clients are configured with list)
- Clients contact any alive GMS process and join, maintain keep-alive
  - If client detects GMS process is dead, it attaches to another
  - If GMS process detects client is dead, it removes the client from the group list
- GMS processes maintain group lists, report changes to clients
Recall *sequencer* and *token* approaches to ordering

Sequencer-based Ordering

Token-based Ordering

Analogous approaches to membership views

- Organizer-based membership
  - ISIS, Ensemble
- Token-based membership
  - Totem, Transis
- We’ll look at Totem and ISIS
Token-based membership

- Based on a single message: JOIN
- Used by newly joining process
- Also used by a process that detects a failure in another process
  - For instance, the process expecting to get the token next
- JOIN message contains:
  - List of included processes (those in the new view)
  - List of excluded processes (those not in the new view)

Basic algorithm

- Every process periodically broadcasts JOIN until it hears identical JOINs from all other included processes
Token membership:
process crashed

Token membership:
process didn’t crash
Token membership: new process joins during algorithm

Token membership state transition diagram
### Token membership states

- **Operational**—normal message delivery (stable ring)
- **Gather**—JOIN messages
- **Commit**—Uses token to verify agreement on membership
  - Required because JOIN messages may be received out of order
  - Token initiated by lowest-ID process
  - Token travels around ring twice—like two-phase commit (2PC)

---

### Token membership states

- **Recovery**—Before recovering messages, must recover token sequence number, retransmit list, and ARU
  - Token was lost in previous “ring” (view), so token contents also lost
  - This probably done during commit
  - Recovery finished when retransmit list is empty, and ARU = token sequence number
  - Token itself indicates final transition to **Operational** state
Commit and Recovery

Next lecture we’ll look at Coordinator-based membership