State machine replication

- We want high scalability and high availability
  - Achieve via redundancy
- High availability means replicated functioning components will take place of ones that stop working
  - Active-passive: replicated components are standing by
  - Active-active: replicated components are working
- Model system as a sequence of states
  - Input to a specific state produces deterministic output and a transition to a new state
  - “State”: replicated data or replicated computing
  - To ensure correct execution & high availability
    - Each process must see & process the same inputs in the same sequence
    - Obtain consensus at each state transition

Faults

- Faults may be
  - Fail-silent (fail-stop)
  - Byzantine (corrupted data)

- synchronous system vs. asynchronous system
  - Synchronous = system responds to a message in a bounded time
  - E.g., IP packet versus serial port transmission
  - We assume we have an asynchronous system

Agreement in faulty systems

- Two army problem
  - Good processors
  - Asynchronous & unreliable communication lines
  - Coordinated attack
  - Infinite acknowledgement problem
Agreement in faulty systems

• It is impossible to achieve consensus with asynchronous faulty processes
  – There is no way to check whether a process failed or is alive but not communicating (or communicating quickly enough)

• We have to live with this

• We cannot reliably detect a failed process

• But we can propagate our knowledge that we think it failed
  – Take it out of the group

Group View

• Set of processes currently in the group
• A multicast message is associated with a group view
• Every process in the group should have the same view

• View change
  – When a process joins or leaves the group, the group view changes
  – View change
    • Multicast message announcing the joining or leaving of a process

Virtual Synchrony

• What if a message is being multicast during a view change?
  – Two multicast messages in transit at the same time:
    • view change (vc)
    • message (m)

• Need to guarantee
  – m is delivered to all processes in G before any process is delivered vc
  – OR m is not delivered to any process in G

• Reliable multicasts with this property are virtually synchronous
  – All multicasts must take place between view changes
  – A view change is a barrier

Virtual Synchrony: implementation example

• Isis: fault-tolerant distributed system offering virtual synchrony
  – Achieves high update & membership event rates
  – Hundreds of thousands of events/second on commodity hardware as of 2009

• Virtual synchrony
  – Provides distributed consistency
  – Applications can create & join groups & send multicasts
  – Applications will see the same events in an equivalent order
  – Group members can update group state in a consistent, fault-tolerant manner

• Who uses it?
  – Isis: Microsoft’s scalable cluster service, IBM’s DCS system, CORBA
  – Similar models: Apache Zookeeper (configuration, synchronization, and naming service)

Implementation: Goals

• Message transmission is asynchronous
  – Machines may receive messages in different order

• Virtual synchrony
  – Preserve the illusion that events happen in the same order
  – Hold back & deliver to the application in a consistent order.
  – Uses TCP → reliable point-to-point message delivery
  – Multicasting is implemented by sending a message to each group member
  – No guarantee that ALL group members receive the message
  – The sender may fail before transmission ends
Implementation: Group Management

- **Group Membership Service (GMS)**
  - Failure detection service
  - If a process p reports a process q as faulty
  - GMS reports this to every process with a connection to q
  - q is taken out of the process group and would need to re-join
  - Imposes a consistent picture on membership

Implementation: State Transfer

- When a new member joins a group
  - It will need to import the current state of the group
  - State transfer:
    - Contact an existing member to request a state transfer
    - Initialize the replica to that checkpoint state
    - A state transfer is treated as an instantaneous event
  - Problem
    - Guarantee that all messages sent to view Gi are delivered to all non-faulty processes in Gi before the next view change (Gi+1)

Implementation: Receiving all messages

- Make sure each process in Gi has received all messages that were sent to Gi:
  - A sender may have failed
  - There may be processes that will not receive a message m
  - These processes should get m from somewhere else
- Let every process hold m until it knows that all members of Gi received it
  - Once all members received it, m is **stable**
  - Only stable messages can get delivered to applications
  - Select an arbitrary process in Gi and request it to send m to all other processes
  - Delivery within the group is reliable, so this ensures that the message is stable

View change: Gi → Gi+1

- Some process P receives a **view change message**
  - It detected a failure or received a request from a process wanting to join or leave the group
  - P forwards a **copy of any unstable messages** to every process in Gi+1
  - It then marks the message as stable
  - P indicates it no longer has any unstable messages
  - It is ready to transition to view Gi+1 as soon as other processes are ready
  - P multicasts a **flush** message for Gi+1
  - Waits to receive a flush message for Gi+1 from every other process
  - Then switches to the new view Gi+1

View change summary

- Every process will
  - Send any unstable messages to all group members
  - Process received messages that are not duplicates
  - Send a flush message to the group
  - Receive a flush message from the entire group
The end