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 the 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 (or both)
  - 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: the system does not communicate
  - fail-stop: a fail-silent system that remains silent
  - fail-recover: a fail-silent system that comes back online
  - Byzantine: the system communicates with bad data

- synchronous system vs. asynchronous system
  - Synchronous = system responds to a message in a bounded time
  - Asynchronous = no assurance of when a message arrives
    - E.g., IP packet versus serial port transmission
  - IP network = asynchronous

Agreement in faulty systems

- Replicas = group of machines = process group
  - Load balancing (queries can go to any replica)
  - Fault tolerance (OK if some dies; they all do the same thing)
- Important for replicas to remain consistent
  - Need to receive the same messages [usually] in the same order
- What if one of the replicas dies?
  - Then it does not get updates
  - When it comes up, it will be in a state prior to the updates
    - Not good – getting new updates will put it in an inconsistent state

Faults

- Faults may be
  - fail-silent: the system does not communicate
  - fail-stop: a fail-silent system that remains silent
  - fail-recover: a fail-silent system that comes back online
  - Byzantine: the system communicates with bad data

- synchronous system vs. asynchronous system
  - Synchronous = system responds to a message in a bounded time
  - Asynchronous = no assurance of when a message arrives
    - E.g., IP packet versus serial port transmission
  - IP network = asynchronous

- We assume we have an asynchronous system
Agreement in faulty systems

- It is impossible to achieve consensus with asynchronous faulty processes
  - There is no foolproof 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
  - Moreover, the system might recover
- But we can propagate knowledge that we think it failed
  - Take it out of the group (even if it is alive)
  - If it recovers, it will have to re-join

Virtual Synchrony

Virtual Synchrony is a software model

Model for group management and group communication
- A process can join or leave a group
- A process can send a message to a group
  - Ordering requirements defined by programmer

Atomic multicast
- "Either all processes received a message or none have"

Group View

- 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
  - Timeouts lead to failure detection
  - Group membership change
    - The dead member is removed from the group

Events

- Group members receive events
  - New message received
  - Group membership change
  - Checkpoint request
    - Dump the state of your system so a new process can read it

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 “all or nothing” semantics
    - 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

Read the distinction between receiving a message and delivering it to the application
**View Changes & Virtual Synchrony**

- **G** = \{p\}
- **G** = \{p, q\}
- **G** = \{p, q, r, s, t\}
- **G** = \{r, s, t\}

**Time**

- 0
- 10
- 20
- 30
- 40
- 50
- 60
- 70

**Virtual Synchrony: implementation example**

- Isis toolkit: 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
- 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?
  - New York Stock Exchange, Swiss Exchange, US NAVY AEGIS, etc.
  - 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 (e.g., IP)
  - Machines may receive messages in different order
- Virtual synchrony
  - Preserve the illusion that events happen in the same 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 of membership to all members

**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 → checkpoint request
    - Initialize the new member (replica) to that checkpoint state
    - A state transfer is treated as an instantaneous event
- Important – enforce the group view barrier
  - Guarantee that all messages sent to \( G_i \) are delivered to all non-faulty processes in \( G_i \) before the next view change \( (G_{i+1}) \)

**Ensuring all messages are received**

- All messages sent to \( G_i \) must be delivered to all non-faulty processes before a view change to \( G_{i+1} \)
- But what if the sender failed?
  - Each process stores a message until it knows all members received it
  - At that time, the message is stable
View change: \( G_i \rightarrow G_{i+1} \)

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

View change summary

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

The End