Fault-tolerance techniques

RSM, Paxos

Jinyang Li
What we’ve learnt so far

• Fault tolerance
  – Recoverability
    • All-or-nothing atomicity for updates involving a single server.
  – 2P commit
    • All-or-nothing atomicity for updates involving >=2 servers.
  – However, system is down while waiting for crashed nodes to reboot

• This class:
  – Ensure high availability through replication
Achieving high availability using replication

Idea: upon A’s failure, serve requests from either B or C.
Challenge: ensure sequential consistency across such reconfiguration
RSM: Replicated state machine

- RSM is a general replication method
  - Lab 8: apply RSM to lock service
- RSM Rules:
  - All replicas start in the same initial state
  - Every replica apply operations in the *same* order
  - All operations must be deterministic
- All replicas end up in the same state
Strawman RSM

- Does it ensure sequential consistency?
RSM based on primary/backup

• Primary/backup: ensure a single order of ops:
  – Primary orders operations
  – Backups execute operations in order
RSM: read-only operations

- Read-only operations need not be replicated
RSM: read-only operations

- Can clients send read-only ops to any server?
RSM: read-only operations

- Can clients send read-only ops to any server?

X’s initial value is 0
RSM failure handling

- If primary fails, one backup acts as the new primary
- Challenges:
  1. How to reliably detect primary failure?
  2. How to ensure no 2 backups simultaneously become primary?
  3. How to preserve sequential consistency across primary changes?
     - Primary can fail after sending an operation $W$ to backup A but before sending $W$ to B
     - A and B must agree on whether $W$ is reflected in the new state after reconfiguration

Paxos, a fault-tolerant consensus protocol, addresses these challenges
Case study: Hypervisor
[Bressoud and Schneider]

• Goal: fault tolerant computing
  – Banks, NASA etc. need it
  – In the 80s, CPUs are very likely to fail
• Hypervisor: primary/backup replication
  – If primary fails, backup takes over
  – Caveat: assuming perfect failure detection
Hypervisor replicates at VM-level

- Why replicating at VM-level?
  - Hardware fault-tolerant machines are big in 80s
  - Software solution is more economical
  - Replicating at O/S level is messy (many interfaces)
  - Replicating at app level requires programmer efforts

- Primary and backup execute the same sequence of machine instructions
A Strawman design

- Two identical machines
- Same initial memory/disk contents
- Start execution on both machines
- Will they perform the same computation?
Hypervisor’s basic plan

- Execute one instruction at a time using primary/backup
Hypervisor Challenges

• Operations must be deterministic.
  – ADD, MUL etc.
  – Read memory (?)

• How to handle non-deterministic ops?
  – Read time-of-day register
  – Read disk
  – Interrupt timing
  – External input devices (network, keyboard)

• Executing one instruction at a time is VERY SLOW
Handle disk operations

Strawman replicates disks at both machines

**Problem:** disks might not behave identically (e.g. fail at different sectors)

Hypervisor connects devices to both machines

- Only primary reads/writes to devices
- Primary sends read values to backup
- Only primary handles interrupts from h/w
- Primary sends interrupts to backup
Hypervisor executes in epochs

- **Challenge**: executing one instruction at a time is slow
- Hypervisor executes in epochs
  - CPU h/w interrupts every N instructions (so both nodes stop at the same point)
  - Primary delays all interrupts till end of an epoch
  - Primary sends all interrupts to backup
  - Primary/backup execute all interrupts at an epoch’s end.
Hypervisor failover

• Primary fails at epoch E
  – backup times out waiting for primary to announce end of epoch E
  – Backup delivers all buffered interrupts at the end of E
  – Backup starts epoch E+1
  – Backup becomes primary at epoch E+1

• What about I/O at epoch E?
Hypervisor failover

• Backup does not know if primary executed I/O epoch E?
  – Relies on O/S to re-try the I/O
• Device needs to support repeated ops
  – OK for disk writes/reads
  – OK for network (TCP will figure it out)
  – How about keyboard, printer, ATM cash machine?
Hypervisor implementation

- Hypervisor needs to trap every non-deterministic instruction
  - Time-of-day register
  - HP TLB replacement
  - HP branch-and-link instruction
  - Memory-mapped I/O loads/stores
- Performance penalty is reasonable
  - A factor of two slow down (HP 9000/720 50MHz)
  - How about its performance on modern hardware?
Caveats in Hypervisor

• Hypervisor assumes failure detection is perfect
• What if the network between primary/backup fails?
  – Primary is still running
  – Backup becomes a new primary
  – Two primaries at the same time!
• Can timeouts detect failures correctly?
  – Pings from backup to primary are lost
  – Pings from backup to primary are delayed
Paxos: fault tolerance consensus
Paxos: fault tolerant consensus

- Paxos lets all nodes agree on the same value despite node failures, network failures and delays
- Extremely useful:
  - e.g. Nodes agree that X is the primary
  - e.g. Nodes agree that W should be the most recent operation executed
Requirements of consensus

- **Correctness (safety):**
  - All nodes agree on the same value
  - The agreed value X has been proposed by some node

- **Fault-tolerance:**
  - If less than some fraction of nodes fail, the rest should still reach agreement

- **Termination:**
Fischer-Lynch-Paterson [FLP’85] impossibility result

• It is impossible for a set of processors in an asynchronous system to agree on a binary value, even if only a single processor is subject to an unannounced failure.

• Asynchrony --> timeout is not perfect
Paxos

- Paxos: the only known fault-tolerant agreement protocol
- Paxos’ properties:
  - Correct
  - Fault-tolerance:
    - If less than $N/2$ nodes fail, the rest nodes reach agreement *eventually*
  - No guaranteed termination
Paxos: general approach

- One (or more) node decides to be the leader
- Leader proposes a value and solicits acceptance from others
- Leader announces result or try again
Paxos’ challenges

- What if >1 nodes become leaders simultaneously?
- What if there is a network partition?
- What if a leader crashes in the middle of solicitation?
- What if a leader crashes after deciding but before announcing results?
- What if the new leader proposes different values than already decided value?
Paxos setup

• Each node runs as a *proposer, acceptor* and *learner*

• Proposer (leader) proposes a value and solicit acceptance from acceptors

• Leader announces the chosen value to learners
Strawman

• Designate a single node $X$ as acceptor (e.g. one with smallest id)
  – Each proposer sends its value to $X$
  – $X$ decides on one of the values
  – $X$ announces its decision to all learners

• Problem?
  – Failure of the single acceptor halts decision
  – Need multiple acceptors!
Strawman 2: multiple acceptors

- Each proposer (leader) propose to all acceptors
- Each acceptor accepts the first proposal it receives and rejects the rest
- If the leader receives positive replies from a majority of acceptors, it chooses its own value
  - There is at most 1 majority, hence only a single value is chosen
- Leader sends chosen value to all learners

Problem:
- What if multiple leaders propose simultaneously so there is no majority accepting?
- What if the leader dies?
Paxos’ solution

• Each acceptor must be able to accept multiple proposals

• Order proposals by proposal #
  – If a proposal with value v is chosen, all higher proposals have value v
Paxos operation: node state

- Each node maintains:
  - $n_a$, $v_a$: highest proposal # accepted and its corresponding accepted value
  - $n_h$: highest proposal # seen
  - $my_n$: my proposal # in the current Paxos
Paxos operation: 3P protocol

• Phase 1 (Prepare)
  – A node decides to be leader (and propose)
  – Leader choose \( m_n > n_h \)
  – Leader sends \(<prepare, m_n>\) to all nodes
  – Upon receiving \(<prepare, n>\)
    - If \( n < n_h \)
      - reply \(<prepare-reject>\)
    - Else
      - \( n_h = n \)
      - reply \(<prepare-ok, n_a,v_a>\)
    
      This node will not accept any proposal lower than \( n \)
Paxos operation

• Phase 2 (Accept):
  – If leader gets prepare-ok from a majority
    \( V = \) non-empty value corresponding to the highest \( n_a \) received
    If \( V = \) null, then leader can pick any \( V \)
    Send \(<\text{accept}, my_n, V>\) to all nodes
  – If leader fails to get majority prepare-ok
    • Delay and restart Paxos
  – Upon receiving \(<\text{accept, n, V}>\)
    If \( n < n_h \)
      reply with \(<\text{accept-reject}>\)
    else
      \( n_a = n; v_a = V; n_h = n \)
      reply with \(<\text{accept-ok}>\)
Paxos operation

• Phase 3 (Decide)
  – If leader gets accept-ok from a majority
    • Send <decide, v_a> to all nodes
  – If leader fails to get accept-ok from a majority
    • Delay and restart Paxos
Paxos operation: an example

\[\begin{align*}
\text{nh}=\text{N0:0} & \quad \text{na} = \text{va} = \text{null} \\
\text{nh}=\text{N1:0} & \quad \text{na} = \text{va} = \text{null} \\
\text{nh}=\text{N2:0} & \quad \text{na} = \text{va} = \text{null}
\end{align*}\]

\[\begin{align*}
\text{nh}=\text{N1:1} & \quad \text{na} = \text{null} \\
\text{nh}=\text{N1:1} & \quad \text{na} = \text{N1:1} \\
\text{nh}=\text{N1:1} & \quad \text{na} = \text{N1:1}
\end{align*}\]

\[\begin{align*}
\text{nh}=\text{N1:1} & \quad \text{na} = \text{va} = \text{null} \\
\text{nh}=\text{N1:1} & \quad \text{na} = \text{va} = \text{null} \\
\text{nh}=\text{N1:1} & \quad \text{na} = \text{va} = \text{null}
\end{align*}\]
Paxos properties

• When is the value V chosen?
  1. When leader receives a majority prepare-ok and proposes V
  2. When a majority nodes accept V
  3. When the leader receives a majority accept-ok for value V
Understanding Paxos

• What if more than one leader is active?
• Suppose two leaders use different proposal number, N0:10, N1:11
• Can both leaders see a majority of prepare-ok?
Understanding Paxos

• What if leader fails while sending accept?
• What if a node fails after receiving accept?
  – If it doesn’t restart …
  – If it reboots …
• What if a node fails after sending prepare-ok?
  – If it reboots …
Using Paxos for RSM

- Fault-tolerant RSM requires consistent replica membership
  - Membership: <primary, backups>

- All active nodes must agree on the sequence of view changes:
  - <vid-1, primary, backups><vid-2, primary, backups> ...

- Use Paxos to agree on the <primary, backups> for a particular vid
  - Many instances of Paxos execution, one for each vid.
  - Each Paxos instance agrees to a single value, e.g. v[1]=x, v[2]=y, ...
Lab7: Using Paxos to track view changes

All nodes start with static config vid1:N1, Paxos-instance-1 has static agreement v[1]:N1

N2 joins

V[2]: N1,N2

Paxos-instance-2: make N1 agree on v[2]

N3 joins

V[3]: N1,N2, N3

Paxos-instance-3: make N1,N2 agree on v[3]

N3 fails

V[4]: N1,N2

Paxos-instance-4: make N1,N2,N3 agree on v[4]
Lab 7: Using Paxos to track view changes

V[1]: N1
V[2]: N1, N2

Prepare, i=2, N3: 1
oldview, v[2]=N1, N2

V[1]: N1
N3 joins

V[1]: N1
V[2]: N1, N2

N1
N2
N3
Lab8: Using Paxos to track view changes

V[1]: N1
V[2]: N1,N2

V[1]: N1
V[2]: N1,N2

Prepare, i=3, N3:1

N3 joins

V[1]: N1
V[2]: N1,N2
Lab8: reconfigurable RSM

• Use RSM to replicate lock_server
• Primary (master) assigns a viewstamp to each client requests
  – Viewstamp is a tuple (vid:seqno)
• Primary can send multiple outstanding requests to backups
  – All replicas execute client requests in viewstamp order
Lab8: Reconfigurable RSM

• What happens during a view change?
  – The last couple of outstanding requests might be executed by some but not all replicas

• Must sync the state of all replicas before accepting requests in new view
  – All replicas transfer state from the primary
  – Since all agree on the primary, all replicas’ state are in sync