Cloud Controllers

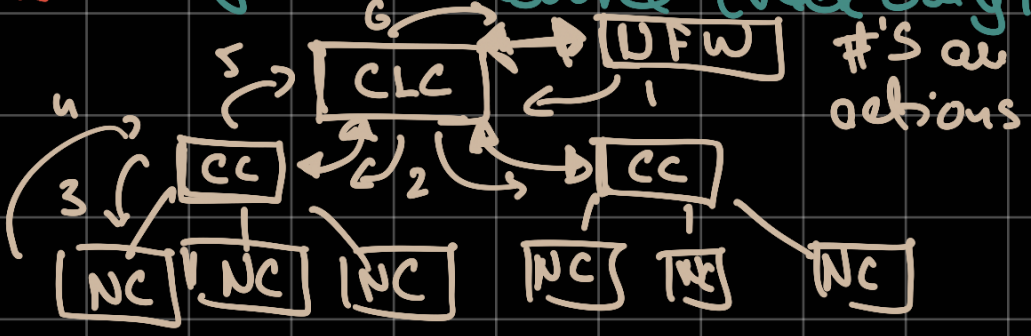
① Eucalyptus : Opensource designed to build pvt or hybrid clouds. Compatible with EC2 or EC3 instances

UFW: User facing web service : Accepts Aws Compatible API

Cloud Controller : High level resource track only!

Node "

Storage "



Distributed Systems

•> Arch Model → System Arch (placement/responsibility)
→ Sw Arch (organisation of Sw)

Interaction model (Comms)

fault model (failure modes & steps to recovery)

•> 2 main models: P2P & H-S

•> AWS Aurora - RDBMS, Multi leader P2P impl.

(*) Troubles with dist Sys

•> The non deterministic & possibility of "partial failures" make dist Sys hard.

•> Reliability is the probability that a system will meet certain perf req & yield correct output for a specific time!

•> Possible faults:

Transient faults | Intermittent | Permanent

MTBF = total uptime / # of breakdowns

•> How long should time out be? 2d+r? most networks can't promise this work:

= Async comm for no timeouts

= experimentally choose "

•> Detection failure in dist sys:

- Heartbeats (C→S) • Probing (S→C)

GFS

if chunk server misses beat starts replicating.

•> On fault detection: - Async comms - exponential backoff

- Work around net timeouts.
- Circuit breaker pattern.
- limit # of queued req.
- fallbacks

•> recovery modes: Active-Active | Active-passive.

↳ VMWare FT (2 VMs)

(*) Performance metrics

% availability/uptime = $\frac{\text{agreed time} - \text{down time}}{\text{agreed time}}$ of

down time = $\frac{\text{down time}}{\text{agreed time}}$

MTBF = $\frac{\text{agreed time}}{\#\text{ of failures}}$ MTTF = $\frac{\text{Correct operational time}}{\#\text{ of fails}}$

MTTR = $\frac{\text{downtime}}{\#\text{ of failures}}$

Say 3 System fail at 10, 12 & 14 : MTTF = $\frac{10+12+14}{3} = 12$

System availability : $\frac{MTTF}{MTTF + MTTR}$ (3.65 days/year) 99%

→ Consensus problem in Dist Sys

•> We consider all system to be simply "State machines": disty sys are "replicated state machines"

•> But this is made hard due to problems like:
- faulty nodes - Unreliable network
- lack of sync global clock.

•> Consensus is achieved if:
- Agreement (common output) - Termination (eventually some output) - Validity
- Integrity (one node only pitches in once) - Non-triviality.

•> Things to consider:
- Unreliable multicast - membership failure
- leader election - exclusion of member

-> We can have Synchronous Consensus (not practical)
On async (not strongly consistent)

(*) Paxos algorithm

-> 3 roles: Proposers, acceptors, learners.
CANT HANDLE non-BFT faults

-> Safety: only proposed values can be accepted
Only single value
Unless accepted no assumptions.

-> If proposer selects a number "n" & sends request to acceptor. If acceptor has not accepted a number higher than n, then it "can" accept n.
If propose gets majority (ready) acks from acceptors. It can send back an (accept) signal.
An acceptor accepts on signal unless higher #.

This can lead to infinite proposals if time to proposal is less than time from proposal to accepted.

Sdu: Distinguished proposer!

Leader election in Dist Sys

·) elect 1 leader at a time (Safety)

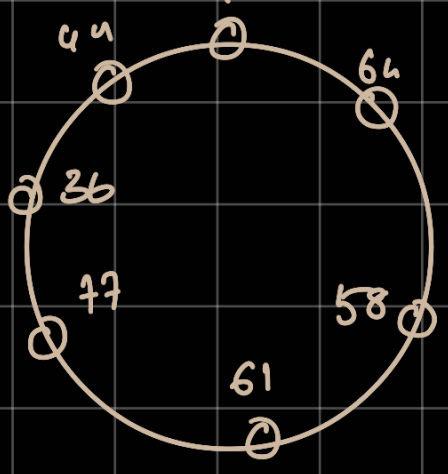
every node is eventually either elected or not (liveness)

↳ => there will be leaders eventually to move the system fwd

(*) Ring leader election

Algorithm:

:> i. N node comm only with neighbour



ii. If any node thinks leader failed,
it sends a id: attr for election

iii. Any node that recieve it either:
fwds if id > own id if not
Overwite with own id.

iv. If id = own id, this id must be greatest, it then send an "elected" message around.

v. On elected message, it records elected id & fwds till until elected id ≠ Self id.

Complexity = 3N-1
O(N)

hence

Doesn't not satisfy liveness if failure happens during election.

(*) Modified ring leader

·) Algorithm: i) Any failed nodes are bypassed. nodes have meshed comms.

ii) Each node "Appends" its id to the msg & propagate

iii) when it reaches the initator, it selects the best node. It now sends a Coordinator message with new id, each node appends if to be elected node id is true twice it is. This way it handles during election fails.

Number/Complexity: 2N O(N)

(*) Bully Algorithm

-> when node with the second highest id detects failure it elects itself.

↳ Sends Co-ord msg to all lower ids

-> election:

If it knows to be highest - elected.

else If Sends election msg to nodes with higher id.

If no answer elects itself. if there is a response wait for co-or on timeout.

Messages overhead: $O(N^2)$

Distributed Locking

-> distributed locks are mutal exclusion locks that have a global presence.

-> locking items to gain access often does more harm in a dist sys. DONT DO IT.

•) locking although helps resolve:
→ only one node access → efficiency (no retries)
→ Correctness.

•) Types of dist locks: - optimistic - pessimistic
(we dont block) ("Absolute control")
dangerous step

Upgrading old read locks to write locks! ⇐ allowing for multiple write locks.

•) fencing: rejecting actions from timedout locks using token #'s
If the system has issues a lock with higher token all lower token are invalidated & fenced.

•) DLM: distributed lock manager : a common replicated database in every node. This DB controls the lock to shared resources.
eg: Zookeeper, Google chubby lock, Redis ??!

(*) Zookeeper

•) It serves as a drop in solution for cluster co-ord in large clusters, It:
- Supports use of locks where needed
- Ensures atomicity of dist txns.

- Helps maintain Namespaces
- HA, HT, Low latency
- Tuned for reads

- Uses "Znodes" to maintain Namespaces, provides the following on top (inside) Znode:
  - Naming
  - Config management
  - Lock & Sync
  - Group Service

- Each application can create it Znode & updates it with stats. All Znodes can observe this.
- Zk in itself maintains a (replica of znode ensemble) Servers with one acting as leader.
- Znode maintain hierarchial NS path (all in mem)
- Types of Znode:
  - i) ephemeral nodes: Session based, useful for detecting process termination. CANT have children.
  - ii) Sequence nodes: Append a monotonically increasing counter to end to each child.
- Each Znode can store only KB's of data.

Znode stores:
- timestamps
- Acl changes
- transactions
- version #'s

CAS (oldver, value) Compare & Set.

each transaction is unique with Zxid.

-> ZK actions:
- read: blw CS
- write: blw S-S for consensus & later C-S
- watch: S-S → always funded to leader.

-> ZK API:
- Create - delete - exists - get data
- Set data - get child - Sync (waits for data prop)
- get ACL - set ACL

-> Adv:
- Sync - reliability - Atomicity

Dadv:
- Complex - No discovery