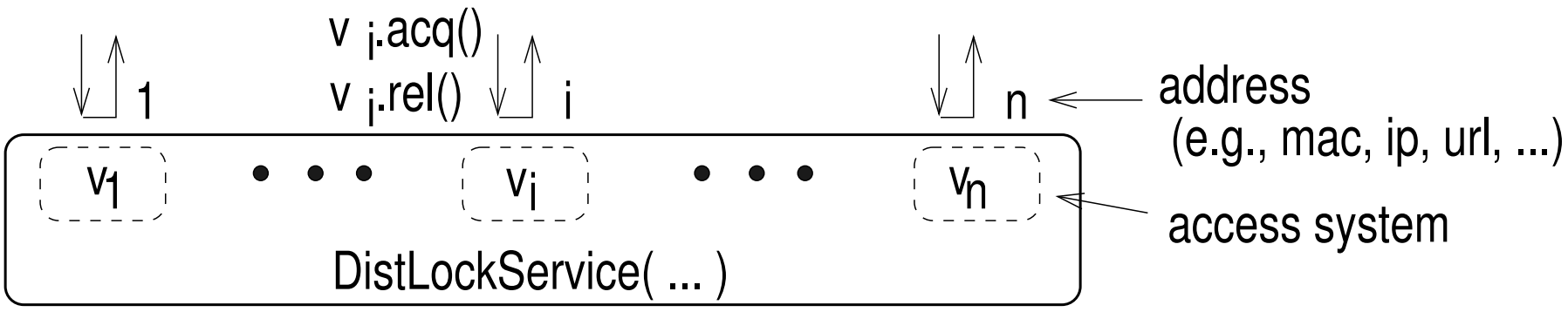
Distributed Lock Service

Shankar

April 18, 2014

Overview: Distributed Lock Service



- Parameter: set of addresses
- Access system at each address
  - sid returned at instantiation
- Functions at address j, assuming access system v_j
  - input v_j.acq(): acquire lock
  - input v_j.rel(): release lock
- Termination function(s): can be added

Service DistLockService(ADDR) – 1

- Main
  - ic {ADDR not empty }
  - eating $\leftarrow$ null // user with lock if not null
  - users$_j$ $\leftarrow$ [] // users at addr j
  - v$_j$ $\leftarrow$ sid() // sid of access system at j
  - return {v$_j$} // map of sids
- atomicity assumption: input and output parts
- progress assumptions
  - thread u in v$_j$.rel *leads-to* not u in v$_j$.rel
  - (eating $\neq$ null *leads-to* eating $=$ null) $\Rightarrow$
    (u in users$_j$ *leads-to* u $=$ eating)

Service DistLockService(ADDR) – 2

- v[j].acq()
  - ic {eating $\neq$ mytid}
    add mytid to users_j
  - oc {eating $=$ null}
    eating $\leftarrow$ mytid

- input v[j].rel()
  - ic {eating $=$ mytid and
    mytid in users_j} // caller acquired lock at j
    remove mytid from users_j
  - oc {true}

Inverse of distributed lock service

- DistLockServiceInverse(ADDR, v)
  - main: ... ~~v_j~~ ...
  - output doAcq(j) ~~input v_j.acq()~~
    - ~~ic~~ oc {...} ... v_j.acq()
    - ~~oc~~ ic {...} ...
  - output doRel(j) ~~input v_j.rel()~~
    - ~~ic~~ oc {...} ... v_j.rel()
    - ~~oc~~ ic {...} ...
  - atomicity assumption: ...
  - progress ~~assumption~~ condition {...}

Some naive implementations over fifo channel

- Centralized solution
  - fixed access system, say v_0 controls lock
  - acq and rel calls send msgs to v_0
  - v_0 serves acq-call msgs in fifo order
- Token-circulating solution
  - "token" msg cycles through access systems
  - when an access system gets token
    - if local hungry user
      return an acq call; wait for rel call
    - forward token to next access system
- Ideal solution
  - request disturbs only non-thinking access systems
  - distributed path-reversal solution // chapter 16