

ARCHITECTURE DESIGN

Nome del programma: **CLUSTER !! TEMPORANEO !!**

2 PROGRAMMI: main-server (assegnato da desktop VM)
client,

node-creation,

load-balancer (stupido e banale)

POSSIBILI LINGUAGGI: RUST, GO

i programmi operano in locale su una stessa macchina,
i nodi saranno delle VM, inizialmente 3 per
ridondanza.

I nodi potranno essere aggiunti tramite
istanze del programma node-creation.

VM:



17A|W SERVER:

```
VAR LOG =  $\emptyset$ 
```

```
main() {
```

```
    input_messages();
```

```
}
```

$TIMEOUT-N = \text{COMMON} + \text{RANDOM}$

```
fun input_messages {
```

```
    While (TRUE) do
```

```
        recvr ( APPENDENTIZY m).TIMEOUT (TIMEOUT-N)
```

```
        • OK ( async read_message(m, sender) )
```

```
        • FAIL ( async indic_election() );
```

```
    done
```

```
}
```

```
fn indice_elezioni()
```

```
{
```

```
    status.increase_term();
```

```
    shell(request_vote);
```

```
}
```

Num update-index = 0

Num Voted-for = nil

fn read-message(m, sender)

switch type-message(m){

case z-e: // APPEND ENTRY
COROUTINE(append-entry-mex(m, sender));
break;

case req-v: // REQUEST VOTE
COROUTINE(other-node-vote-candidate(m, sender));
break;

case new-c: // new client connection
COROUTINE(input-data-user(m, sender));
break;

case zcc-c:
COROUTINE(add-supporter(m, sender));
break;

case lb-l: // Load-balancer-leader
COROUTINE(answer-load-balancer(m, sender));
break;

case l-new-conf: // external to leader
COROUTINE(new-conf(m, sender));
break;

case f-new-conf: // new-node in cluster
COROUTINE(copy-state(m, sender));
break;

```

fn append_entry_max(m, sender) {
    state = follower
    if (check_consistency(v.e.prev_log_index,
                        v.e.prev_log_term))
    then
        send(leader, {TRUE,
                      my_term});

        update_state(v.e.entries, prev_log_index);
        update_index(v.e.leader_commit);

    else
        send(sender, {FALSE, my_term});

}

```

Leader \rightarrow Bool leader [become_leader]

```

fn answer_load_balancer()
{
    if (Leader)
        send(sender, true);
    else
        send(sender, false);
}

```

```

Bool vote → add_entry.vote
fn other_node_vote_candidate (m, sender)
{
  if (not(vote)) then return endif;
  if (m.term < my_term) then send(sender, my_term, false) endif
  if ( !more_recent_log(m.last_log_index, m.last_log_term) )
  then
    send(sender, my_term, false)
  else if (already_vote = nil || already_vote = sender)
  then
    send(sender, my_term, true); already_vote = sender;
  else
    send(sender, my_term, false)
  endif
}

```

Bool leader = false

```

fn become_leader()

```

```

{
  send_all (APPEND-ENTRY);
  leader = true;

```

While true

do

```

  send_all (APPEND-ENTRY);
  wait(timeout);

```

done

```

}

```

$\text{voted_for} \rightarrow \text{voted_for}[\text{accept_vote}]$

$n_nodes_in_cluster = C$

$n_supporter = 0$

$n_non_supporter = 0$

```
{  
  add_supporter(m, sender)  
}
```

```
  if (m.vote == TRUE)  
    then
```

```
      n_supporter ++;
```

```
  else
```

```
      n_non_supporter ++;
```

```
  endif
```

$\text{var } n_victory = (n_nodes_cluster / 2)$

```
  if (n_supporter > n_victory)  
    then
```

```
      become_leader(); voted_for = null;
```

```
  endif
```

```
  if (n_supporter + n_non_supporter == n_nodes_cluster)  
    then
```

```
      voted_for = null
```

```
  endif
```

```
}
```

Bool leader \longrightarrow Recv - leader . leader

{
 input_data_user (m, sender) // max user instr
}

{USER_INSTR, R-W, ENTRY-N, DATA}

if (not (leader)) then

 var leader_id = status.get_leader_ip();

 send (leader_ip, m)

endif

switch (m. R-W) {

 case R:

 FILE = status.get_entry (m.ENTRY-N)

 send (sender, read (FILE))

 break;

 case W:

 data = m.data;

 status.upload_file (data)

 break;

}


```
fn added_node()
{
```

```
    send {MY_ID: IP, NEW_CONF: TRUE,
          TERM: MY_TERM,
          L_1: MY_LAST_LOG_INDEX,
          L_T: MY_LAST_LOG_TERM, } ;
```

```
}
```

```
Bool leader  $\rightarrow$  become_leader. leader
fn answer_leader_proposer(m, sender)
{
```

```
    send(sender, {LEADER: leader});
```

```
}
```

```
Bool vote := false
```

```
fn copy_state(m, sender)
{
```

```
    add_entry(m.entry, m.term, m.index);
    vote = m.vote
```

```
}
```

```
fn update_new_node(m)
```

```
{  
  foreach (LogEntry l : Logs)  
  {
```

```
    mex = {Bony:l ; VOTANTE:FALSE};
```

```
    send(m, ip_new_node, mex);
```

```
  }
```

```
}
```

list_updated_node = "old_nodes"

List updating = \emptyset

```
{ new_conf(m, sender)
}
```

```
if (not (isEmpty(m.to_add)))
then
```

```
    coroutine( sr_nodes(m.to_add, new) );
endif
```

```
if (not (isEmpty(m.to_remove)))
then
```

```
    coroutine( dr_nodes(m.to_remove, del) );
endif
```

```
}
```

updating \rightarrow new_conf.updating

```
{ sr_nodes(to_add, op)
```

```
}
```

```
Add(updating, to_add);
```

```
foreach (node : to_add)
```

```
{
```

```
    if (op = new) then
```

```
        coroutine( update_node(node, m) );
```

```
    elseif (op = DEL) then
```

```
        coroutine( remove_node(node) );
```

```
    }
```

```
fn update_node (node, m)
```

```
  Add_log_entry ("joint_conf.", node)
```

```
  send_all (APPEND-ENTRY);
```

```
  update_new_node (node, m)
```

```
  send ( node , {BODY: NBCL; NOT ANTE=TRUE})
```

```
  Add (list_updated_nodes, node);
```

```
  Remove (updating, node);
```

```
  status.add_log_entry ("nodes updated", m.ip_new_node);
```

```
  if ( not (isEmpty (updating)) ) then return;
```

```
  send_all (APPEND-ENTRY);
```

```
}
```

```
fn remove_node (node)
```

```
{
```

```
  Add (updating, node)
```

```
  Add_log_entry ("removed node:" node)
```

```
  Remove (updating, node);
```

```
  if ( not (IsEmpty (updating)) ) then return;
```

```
    send_all (APPEND-ENTRY);
```

```
}
```

```
fn send_all ( message_type T)
```

```
{
```

```
  switch (T);
```

```
    case REQ-VOTE:
```

```
      coroutine ( send_all_request_vote () );
```

```
      break;
```

```
    case APPEND-ENTRY:
```

```
      coroutine ( send_all_append_run_time () );
```

```
      break;
```

```
}
```

list_updated_nodes \rightarrow New-conf, list_updated_nodes

```
fn send_all_request_vote ()  
{
```

```
var mex = { REQUEST_VOTE :
```

```
status.get_term(), // my-term
```

```
status.get_id(), // my-id
```

```
status.get_ll_index, // my-LAST-LOG-index
```

```
status.get_ll_term // my-LAST-LOG-term
```

```
foreach ( node-ip in list_updated_nodes ) {
```

```
  COROUTINE ( send ( node-ip , mex ), timeout ( TIME )
```

```
    . FAIL ( remove_node ( node-ip ) ) ;
```

```
    . OK ( ) ) ;
```

```
} }
```

list_updated_nodes \rightarrow New-conf, list_updated_nodes

```
fn send_all_append_entry ()  
{
```

```
var mex = { APPEND_ENTRY :
```

```
status.get_term(),
```

```
status.get_id(),
```

```
status.get_prev_ll_index(), //prev-log-index
```

```
status.get_prev_ll_term(), //prev-log-term
```

```
status.get_new_entries(), //uncommitted entries
```

```
status.get_leader_commit() //index last committed  
entries
```

```
foreach ( node-ip in list_updated_nodes ) {
```

```
  COROUTINE ( send ( node-ip , mex ), timeout ( TIME )
```

```
    . FAIL ( remove_node ( node-ip ) ) .
```

```
    . OK ( ) ) ;
```

```
}
```

```
}
```