ARCHITECTURE DESIGN

Nome del programma: CLUSTER !! TEMPORANEO !!

2 PROGRAMMI: main-server (eseguito da deamon 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, inizialment 3 per
rindondanza.
I nodi potranno essere aggiunti trammite
instanze del programma node_creation.

VM:

MAIN SERVER:

```
VAR LOG = ∅

main(){

        input_messages();
}
```

```
TIMEOUT_N = COMMON + RANDOM

fun input_messages {

    While (TRUE) do
                    recv(APPENDENTRY m).TIMEOUT(TIMEOUT_N)
                    .OK(async read_message(m, sender))
                    .FAIL(async indici_elezioni());
        done
}
```

```
fn   indire_elezioni()
{

    my_term ++;
    broadcast message(REQUEST_VOTE :
                          - MY_TERM
                          - MY_ID
                          - MY_LAST_LOG_INDEX
                          - MY_LAST_LOG_TERM
                      ).send()

}
```

```
Num update_index = 0
var Voted_for = NIL

fn read_message(m, sender)
{
        switch type_message(m){

                case a_e: //APPEND ENTRY
                    COROUTINE(append_entry_mex(m,sender));
                    break;
                case req_v : //REQUEST VOTE
                    COROUTINE(other_node_vote_candidature(m,sender))
                    break;
                case new_c : //new client connection
                    COROUTINE(input_data_user(m,sender));
                    break;

                case acc_c :
                    COROUTINE( add_supporter(m,sender));
                    break;
                case lb_l //Load_balancer_leader
                    COROUTINE(answer_load_balancer(m, sender);
                    break;
```

CASI MESSAGGI DAL FOLLOWER AL LEADER.

```
fn  append_entry_mex(m, sender) {
            state = follower
            if(check_consistency(a_e.prev_log_index,
                                          a_e.prev_log_term))

            then
                  send(leader, {TRUE,
                                    MY_TERM});

                  update_state(a_e.entrys, prev_log_index);
                  update_index(a_e.leader_commit);

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

}


 Leader ——> Bool leader [become_leader]
fn  answer_load_balancer()
 {
    if(Leader)
       send(sender, true);
    else
       send(sender, false);

  }
```

```
fn    other_node_vote_candidature (m, sender)
 {
    if( m.term < my_therm) 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=nill    || 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()

 {
    broadcast_send (APPEND_ENTRY );
    leader = true;

    While  true
    do
        broadcast_send (APPEND_ENTRY );
        wait(timeout);
    done
  }
```

```
voted-for ⟶ voted_for[accept_vote]
n_nodes_in_cluster = C
n_supporter = 0
n_non_supporter = 0
fn   add_supporter(m, sender)
 {
     if (m. vote == TRUE)
     then
          n_supporte ++;

     else
          n_non_supporter ++;

     endif

     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
 }
```

```
fn  input_data_user (m, sender)  //mex user instr
 {
                                   {USER_INSTR, R_W , ENTRY_N, DATA}

     switch (m.R_W){
          case R:
               FILE= get_entry (m.ENTRY_N)
               send (sender, read(FILE))
               break;
```