ARCHITECTURE DESIGN

Nome del programma: CLUSTER !! TEMPORANEO !!

2 PROGRAMMI: main-server (eseguito da demon VM)
client,
node_creation,
load_balancer (stupido e banale)

POSSIBILI LINGUAGGI: RUST, GO

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

I nodi potranno essere aggiunti tramite
istanze del programma node_creation.

VM:

MAIN SERVER:

VAR STATO {FOLLOWER, LEADER, NON_VOTANTE, UNKNOWN} = UNKNOWN

VAR LOG = ∅

```
Main(){
    COROUTINE ( input_messages() );
    COROUTINE ( input_data_user() );

    wait(∞) //while true { }
}
```

TIMEOUT-N: COSTANTE + RANDOM

```
fun input_messages {

  While (TRUE) do
        recv ( APPEND ENTRIES m).TIMEOUT (TIMEOUT_N)
        • OK ( async read_message(m));
        • FAIL ( async indici_elezioni())

  done
}
```

```
fn indice_elezioni()
{
    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 {

    switch type_message(m){

        case    a_e ;     //HEARTBIT LEADER
            state = follower
            if(check_consistency(a_e.prev_log_index,
                                 a_e.prev_log_term))

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

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

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

            break;

        case    req_v : //REQUEST VOTE
            if((voted_for == NIL) && log_check(req_v.last_log_indx,
                                               req_v.last_log_term)
            && (req_v.term < my_term))
                then
                    send(my_term, FALSE)

                else
                    send(my_term, TRUE)
                endif;
}}
```