Programming Distributed Erlang Applications: Pitfalls and Recipes

Abstract

spawn

2. Intra-Node Programming

2.1 Basic Message Passing Guarantee: Stream Semantics

\[ P \]

\[ m \]

\[ P \]

\[ P \]

\[ P \]

\[ P \]

\[ extract \]

\[ receive \]
4. Pitfalls of Inter-Node Programming

4.1 Pitfall 1: Pid Reuse

3. Inter-Node Programming

run

______

N₁

halt

Pid
#!/bin/sh
NODE=$1
while [ 1 -lt 2 ]; do
erl -sname $NODE
sleep 1
done

Figure 1.
-module(pidReuse).
-export([start/0,run/0,echo/0,communicator/1]).

define(N1,'n1@localhost').
define(N2,'n2@localhost').

start() ->
    spawn(?N1,?MODULE,run,[]).
run() ->
erlang:process_flag(trap_exit,true),
Pids =
    lists:map(fun(N) ->
        Pid1 = spawn_link(?N2,erlang,self,[]),
        spawn(?N2,erlang,halt,[]),
        timer:sleep(2000),
        Pid1
    end, lists:seq(1,3)),
    lists:foreach(fun(Pid) ->
        receive {'EXIT',Pid,}_ -> ok end
        end,Pids),
    spawn(?N2,?MODULE,echo,[]),
    communicator(Pids).

    receive {From,N} -> From!{self(),(N+1)} end.

communicator(Pids) ->
    lists:foreach(fun(Pid) ->
        io:format("Trying to communicate with: ~w\n(~w)\n", [Pid,term_to_binary(Pid)]),
        Pid!{self(),S},
        receive {Pid2,N} ->
            io:format("Received ~w from ~w\n(~w)\n", [N,Pid2,term_to_binary(Pid2)]),
            after 2000 ->
                io:format("No reply!\n")
        end,
        Pids).
Figure 2.

Figure 3.
-module(comm).
-export([start/0,snd/2,rcv/0]).
define(N1,'n1@host1.domain.com').
define(N2,'n2@host2.domain.com').

start() ->
    Pid = spawn(?N1,?MODULE,rcv,[]),
    spawn(?N2,?MODULE,snd,[Pid,1]).
rcv() ->
    receive
        N -> io:format("got ~p\n",[N]),
            rcv()
    end.
snd(Pid,N) ->
    Pid!N,
    timer:sleep(1000),
    snd(Pid,N+1).
Figure 4.

Trying to communicate with: <4888.40.0>
(<<131,103,0,12,…,49,56,0,0,0,0,0,0,0,3>>) Recived 6 from <4888.40.0>
(<<131,103,100,0,12,…,49,56,0,0,0,0,0,0,0,3>>) Trying to communicate with: <4888.40.0>
(<<131,103,100,0,12,…,49,56,0,0,0,0,0,0,0,1>>) No reply! Trying to communicate with: <4888.40.0>
(<<131,103,100,0,12,…,49,56,0,0,0,0,0,0,0,2>>) No reply!

4.1.1 Analysis

4.2 Pitfall 2: Misunderstood Basic Communication Guarantees under Distribution
4.2.2 A Revised Message Passing Guarantee

\[
\begin{array}{ccc}
Q & P \\
Q & P & m_1 & m_2 \\
Q & P & Q & Q & P \\
& P & Q & \text{P, } \text{Q} & P \\
\langle m_1 \rangle & \langle m_2 \rangle & \langle m_1, m_2 \rangle
\end{array}
\]

4.3 Pitfall 3: Weaker Multi-Party Communication Guarantees

4.2.1 Analysis

- \( P_1 \rightarrow m_1 \rightarrow P_2 \)
- \( P_1 \rightarrow m_2 \rightarrow P_3 \)
- \( P_3 \rightarrow m_2 \rightarrow P_2 \)

\[
\begin{array}{ccc}
P_1 & m_1 & P_2 \\
P_1 & m_2 & P_3 \\
P_3 & m_2 & P_2 \\
P_2 & m_1 & m_2 \\
P_1 & m_1 & m_2 & P_2 & P_3
\end{array}
\]
4.4 Pitfall 4: Failure Detectors are not Perfect

5. A Programming Discipline for Distributed Applications

5.1 A Simple Programming Discipline

5.2 Possible Extensions to Erlang
Acknowledgement

References

6. Conclusions

```
monitor_node  nodedown
    monitor_node  nodeup
```

[References]


