%%%Ziyang Liu (zl4214) and Hong Lu(hyl14)

Distributed Algorithm Coursework I

Instructions

Six exercises are distributed in six folders: exercise1, exericse2, exercise3, exercise4, exercise5 and exercise6. To run each, type ‘make run1’ in each folder to run the first instruction. Type ‘make run2’ in each folder to run the second instruction, etc.

To set different system parameters, files are copied instead of using command line argument.

System1 - Erlang Broadcast

Submission: show and comment on the output of system1 for message {task1, start, 1000, 3000} followed by message {task1, start, 0, 3000}.

make run1: {task1, start, 1000, 3000};

Output:
<0.30.0>: [{1000,1000},{1000,1000},{1000,1000},{1000,1000},{1000,1000}]
<0.31.0>: [{1000,1000},{1000,1000},{1000,1000},{1000,1000},{1000,1000}]
<0.32.0>: [{1000,1000},{1000,1000},{1000,1000},{1000,1000},{1000,1000}]
<0.33.0>: [{1000,1000},{1000,1000},{1000,1000},{1000,1000},{1000,1000}]
<0.34.0>: [{1000,1000},{1000,1000},{1000,1000},{1000,1000},{1000,1000}]
Comment:
This message causes five process to send 1000 messages to each other, hence receive 1000 messages from each other. Note that the processes rapidly send and receive all the 1000 message before lifetime 3000ms expired.

make run2: {task1, start, 0, 3000};

Output:
<0.30.0>: [{189868,178294},
{189868,175670},
{189868,175876},
{189868,185932},
{189868,189868}]
<0.31.0>: [{185932,178293},
{185932,175670},

{185932,175876},
{185932,185932},
{185932,189867}]
<0.32.0>: [{175876,178293},
{175876,175669},
{175876,175876},
{175876,185932},
{175876,189867}]
<0.33.0>: [{175670,178293},
{175670,175669},
{175670,175876},
{175670,185932},
{175670,189867}]
<0.34.0>: [{178294,178293},
{178294,175669},
{178294,175876},
{178294,185931},
{178294,189867}]
Comment:
This message causes five process to send messages to each other in non-deterministic routes until their time limit exceeds, quite many messages are sent and received rapidly.

System2 - PL Broadcast

Submission: show and comment on the output of system2 for 100 broadcasts with a 1 second timeout followed by an unlimited number of broadcasts (0 Max_messages value) with a 1 second timeout.

make run1: {task1, start, 100, 1000};

Output:
<0.36.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.38.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.40.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.42.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.44.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
Comment:
This message causes five process to send 100 messages to each other, hence receive 100 messages from each other. Still, the processes rapidly send and receive all the 100 message before lifetime 1000ms expired.

make run2: {task1, start, 0, 1000};

Output:
<0.38.0>: [{24459,24436},
{24459,24436},
{24459,24435},
{24459,24434},
{24459,24434}]
<0.40.0>: [{24458,24437},
{24458,24436},
{24458,24436},
{24458,24434},
{24458,24434}]
<0.42.0>: [{24458,24437},
{24458,24436},
{24458,24436},
{24458,24435},
{24458,24434}]
<0.44.0>: [{24458,24437},
{24458,24437},
{24458,24436},
{24458,24436},
{24458,24435}]
<0.36.0>: [{24459,24437},
{24459,24436},
{24459,24435},
{24459,24435},
{24459,24435}]
Comment:
Comparing to System1, a PL is included in the system, which slowed the original system, however, no message is lost here. Less messages are sent and received in a non-deterministic style.

System3 - Best Effort Broadcast

make run1: {task1, start, 100, 1000}

Output:
<0.36.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.39.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.42.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.45.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.48.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]

Comment:
This message causes five process to send 100 messages to each other, hence receive 100 messages from each other. Still, the processes rapidly send and receive all the 100 message before lifetime 1000ms expired although one beb unit is added to the distributed process.

make run2: {task1, start, 0, 1000}

Output1:
<0.39.0>: [{27725,428},{27725,426},{27725,425},{27725,425},{27725,424}]
<0.42.0>: [{27725,428},{27725,426},{27725,425},{27725,425},{27725,425}]
<0.45.0>: [{27724,429},{27724,426},{27724,425},{27724,425},{27724,425}]
<0.48.0>: [{27723,429},{27723,427},{27723,426},{27723,425},{27723,425}]
<0.36.0>: [{27727,428},{27727,426},{27727,425},{27727,424},{27727,425}]
Output2:
<0.42.0>: [{30010,463},{30010,461},{30010,460},{30010,460},{30010,459}]
<0.45.0>: [{30009,464},{30009,461},{30009,460},{30009,460},{30009,460}]
<0.48.0>: [{30008,464},{30008,462},{30008,460},{30008,460},{30008,460}]
<0.36.0>: [{30012,463},{30012,461},{30012,460},{30012,459},{30012,459}]
<0.39.0>: [{30011,463},{30011,461},{30011,460},{30011,459},{30011,459}]

Comment:
The results indicates the system is non-deteministic. Compared with System2, a best effort broadcast is implemented in the system, the communication between BEB and APP slows down the message receiving process. More specifically, the application component in the process needs to wait for the messages passing from be unit to PL link and then back to application. Meanwhile, the application can still keeping sending the beb_broadcast message. Therefore, each process sends a lot of messages but only receive small amount of messages sent by other processes. In addition, compared with the result when Max_message is 100, the process can not keep sending messages so less beb_broadcast request sent from the application component. BEB component can consequently deal with the message queue much more quickly and the application component can receive more message sent by other processes.

We have also considered we need to force the application component to sleep for some time when the application component send too much deliver request but merely small number of messages received. This will consequently reduce the network traffic to allow the application component to receive more messages.

System4 - Unreliable Message Sending

make run1: {task1, start, 100, 1000} with reliability 100

Output:
<0.36.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.39.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.42.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.45.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.48.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]

Comment: This message causes five process to send 100 messages to each other, hence receive 100 messages from each other. Still, the processes rapidly send and receive all the 100 message before lifetime 1000ms expired.

make run2: {task1, start, 100, 1000} with reliability 50

Output:
<0.36.0>: [{100,55},{100,50},{100,60},{100,56},{100,53}]
<0.39.0>: [{100,44},{100,57},{100,58},{100,50},{100,59}]
<0.42.0>: [{100,53},{100,50},{100,52},{100,53},{100,53}]
<0.45.0>: [{100,54},{100,51},{100,56},{100,56},{100,51}]
<0.48.0>: [{100,59},{100,58},{100,49},{100,58},{100,62}]

Comment:
The result reveals that approximately half the number of messages received by the pl link are dropped, This is the exact outcome we expect when the reliability changes from 100 to 50.

make run3: {task1, start, 100, 1000} with reliability 0

Output:
<0.36.0>: [{100,0},{100,0},{100,0},{100,0},{100,0}]
<0.39.0>: [{100,0},{100,0},{100,0},{100,0},{100,0}]
<0.42.0>: [{100,0},{100,0},{100,0},{100,0},{100,0}]
<0.45.0>: [{100,0},{100,0},{100,0},{100,0},{100,0}]
<0.48.0>: [{100,0},{100,0},{100,0},{100,0},{100,0}]

Comment:
The result reveals that all messages received by the pl link are dropped, This is the exact outcome we expect when the reliability changes from 100 to 0.

make run4: {task1, start, 0, 1000} with reliability 100

Output:
<0.36.0>: [{21772,335},{21772,334},{21772,333},{21772,332},{21772,331}]
<0.39.0>: [{21770,335},{21770,334},{21770,333},{21770,332},{21770,331}]
<0.42.0>: [{21771,335},{21771,335},{21771,333},{21771,333},{21771,331}]
<0.45.0>: [{21769,336},{21769,335},{21769,333},{21769,333},{21769,331}]
<0.48.0>: [{21767,336},{21767,335},{21767,334},{21767,333},{21767,332}]

Comment: This result is similar to the result we obtained in system3 when the Max_message is 0 and timeout for all processes is 1 s.

make run5: {task1, start, 0, 1000} with reliability 50

Output:
<0.39.0>: [{23908,509},{23908,507},{23908,509},{23908,515},{23908,497}]
<0.42.0>: [{23895,504},{23895,520},{23895,536},{23895,515},{23895,485}]
<0.45.0>: [{23897,481},{23897,494},{23897,518},{23897,534},{23897,508}]
<0.48.0>: [{23882,496},{23882,549},{23882,515},{23882,500},{23882,511}]
<0.36.0>: [{23896,519},{23896,534},{23896,502},{23896,534},{23896,523}]

make run5: {task1, start, 0, 1000} with reliability 30

Output:
<0.45.0>: [{13446,1262},{13446,1247},{13446,1260},{13446,1262},{13446,1358}]
<0.48.0>: [{13527,1254},{13527,1331},{13527,1252},{13527,1313},{13527,1406}]
<0.36.0>: [{13493,1230},{13493,1281},{13493,1273},{13493,1258},{13493,1313}]
<0.39.0>: [{13457,1235},{13457,1236},{13457,1197},{13457,1261},{13457,1246}]
<0.42.0>: [{13439,1276},{13439,1229},{13439,1188},{13439,1245},{13439,1338}]

make run5: {task1, start, 0, 1000} with reliability 20

Output:
<0.36.0>: [{11868,1758},{11868,2318},{11868,1602},{11868,1647},{11868,1889}]
<0.39.0>: [{13412,1977},{13412,2513},{13412,1863},{13412,2012},{13412,2095}]
<0.45.0>: [{11582,1857},{11582,2404},{11582,1722},{11582,1816},{11582,1998}]
<0.48.0>: [{12297,1756},{12297,2307},{12297,1772},{12297,1674},{12297,1956}]
<0.42.0>: [{11547,1635},{11547,2177},{11547,1621},{11547,1674},{11547,1849}]

make run5: {task1, start, 0, 1000} with reliability 10

Output:

<0.45.0>: [{18890,1928},{18890,1820},{18890,841},{18890,1862},{18890,1775}]
<0.48.0>: [{18403,1876},{18403,1918},{18403,769},{18403,1924},{18403,1880}]
<0.39.0>: [{18811,1963},{18811,1870},{18811,792},{18811,1856},{18811,1861}]
<0.42.0>: [{10045,885},{10045,924},{10045,754},{10045,918},{10045,866}]
<0.36.0>: [{19119,1906},{19119,1829},{19119,804},{19119,1836},{19119,1795}]

Comment for the results above (Reliability 50, 30, 20, 10):
As the reliability decreases, more messages are dropped by the PL unit in the process. This will certainly cause less network deliver requests are sent by PL unit. Therefore, the message queue of the APP component will be smaller and the APP component can have more time to deal with the beb_deliver request. More messages are thus received by all the distributed processes.
Because the message queue of APP component is smaller, App component can finish solving the message queue more quickly in the after 0 clause. More beb_broadcast requests can be sent by App component. More messages can thus sent by each process when reliability decreases.
These two effects mentioned above are combined when the reliability decreases. These two effects make the whole network system much harder to reason. The results above are very interesting. Number of messages sent by processes decrease when the reliability reduces from 50 to 20 and it increase again when the reliability reduces from 20 to 10. We can clearly see the number of received messages increase when the reliability drops especially from 50 to 20.

make run6: {task1, start, 0, 1000} with reliability 0

Output:
<0.39.0>: [{18752,0},{18752,0},{18752,0},{18752,0},{18752,0}]
<0.42.0>: [{18752,0},{18752,0},{18752,0},{18752,0},{18752,0}]
<0.45.0>: [{18752,0},{18752,0},{18752,0},{18752,0},{18752,0}]
<0.48.0>: [{18752,0},{18752,0},{18752,0},{18752,0},{18752,0}]
<0.36.0>: [{18758,0},{18758,0},{18758,0},{18758,0},{18758,0}]

Comment:
The result reveals that all messages received by the pl link are dropped, This is the exact outcome we expect when the reliability changes from 100 to 0.

System5 - Faulty process

make run1: {task1, start, 100, 1000} with process 3 terminates after 5 ms

Output:
<0.42.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.36.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.39.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.45.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.48.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]

Comment:
Although the process 3 is terminated after 5 ms, the process 3 is just able to receive all the messages.

make run2: {task1, start, 0, 1000} with process 3 terminates after 5 ms

<0.42.0>: [{563,10},{563,7},{563,6},{563,6},{563,6}]
<0.45.0>: [{23014,363},{23014,362},{23014,374},{23014,360},{23014,359}]
<0.48.0>: [{23012,363},{23012,362},{23012,374},{23012,360},{23012,359}]
<0.36.0>: [{23018,362},{23018,361},{23018,373},{23018,359},{23018,358}]
<0.39.0>: [{23016,362},{23016,361},{23016,373},{23016,360},{23016,358}]

Comment:
The result clearly show that process 3 is terminated after 5 ms so it can only send and receive very small number of messages. The result is otherwise very similar with system4 when same task is carried out by each process.

System6 - Eager Reliable Broadcast

make run1: {task1, start, 100, 500} with reliability 100, process 3 terminate after 50ms

<0.44.0>: [{100,59},{100,57},{100,56},{100,56},{100,55}]
<0.36.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.40.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]

<0.48.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.52.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
Comment: This output indicates that although process 3 terminates in 50ms, it is still capable of receiving some messages. Generally, our system inclines to send rather than receive.

make run2: {task1, start, 100, 500} with reliability 100, process 3 terminate after 300ms

<0.44.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.36.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.40.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.48.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
<0.52.0>: [{100,100},{100,100},{100,100},{100,100},{100,100}]
Comment:
Comparing to make run1, in make run2, process 3 is capable of receiving all 100 messages.

make run3: {task1, start, 0, 5000} with reliability 100, process 3 terminate after 1000ms

<0.44.0>: [{13288,179},{13288,179},{13288,179},{13288,177},{13288,176}]
<0.36.0>: [{37458,530},{37458,529},{37458,529},{37458,528},{37458,527}]
<0.40.0>: [{37454,531},{37454,531},{37454,530},{37454,529},{37454,528}]
<0.48.0>: [{37447,532},{37447,531},{37447,531},{37447,530},{37447,529}]
<0.52.0>: [{37446,532},{37446,532},{37446,531},{37446,530},{37446,529}]
Comment:
When Max_Messages is set to 0, the system inclines to send message infinitely, which means that PL components are occupied more on sending. Since Erlang is preemptive, PL will have less chance to receive message since there is more message for it to send and the time for a single PL to run should be fixed.

On the other hand of spectrum, it indicates that the S/R message for process 3 is proportional to time it terminates, comparing to other process.

make run4: {task1, start, 0, 5000} with reliability 100, process 3 terminate after 3000ms

<0.44.0>: [{20996,291},{20996,291},{20996,291},{20996,289},{20996,288}]
<0.40.0>: [{37017,525},{37017,524},{37017,524},{37017,523},{37017,522}]
<0.48.0>: [{37010,526},{37010,525},{37010,525},{37010,523},{37010,523}]
<0.52.0>: [{37009,526},{37009,525},{37009,525},{37009,523},{37009,523}]
<0.36.0>: [{37022,524},{37022,523},{37022,523},{37022,521},{37022,521}]

Comment: Process 3 has more S/R messages now.

make run5: {task1, start, 0, 5000} with reliability 30, process 3 terminate after 2000ms

<0.44.0>: [{4667,305},{4667,284},{4667,359},{4667,397},{4667,369}]
<0.48.0>: [{6560,550},{6560,526},{6560,523},{6560,575},{6560,533}]
<0.52.0>: [{6533,542},{6533,465},{6533,482},{6533,557},{6533,519}]
<0.36.0>: [{6544,483},{6544,485},{6544,499},{6544,538},{6544,490}]
<0.40.0>: [{6536,495},{6536,451},{6536,487},{6536,528},{6536,509}]

Comment: It is interesting to see that the sending message decreases quite a bit, however, the the receiving message is similar to the one in make run4.

make run6: {task1, start, 0, 5000} with reliability 50, process 3 terminate after 2000ms

<0.44.0>: [{14017,357},{14017,383},{14017,351},{14017,370},{14017,365}]
<0.48.0>: [{25491,594},{25491,621},{25491,589},{25491,627},{25491,626}]
<0.52.0>: [{25481,602},{25481,632},{25481,578},{25481,586},{25481,625}]
<0.36.0>: [{25504,604},{25504,605},{25504,622},{25504,606},{25504,614}]
<0.40.0>: [{25498,607},{25498,625},{25498,605},{25498,581},{25498,627}]

Comment: When reliability is 50, S/R messages both increases.

make run7: {task1, start, 0, 5000} with reliability 0, process 3 terminate after 2000ms

<0.44.0>: [{14727,0},{14727,0},{14727,0},{14727,0},{14727,0}]
<0.48.0>: [{41895,0},{41895,0},{41895,0},{41895,0},{41895,0}]
<0.52.0>: [{41895,0},{41895,0},{41895,0},{41895,0},{41895,0}]
<0.36.0>: [{41902,0},{41902,0},{41902,0},{41902,0},{41902,0}]
<0.40.0>: [{41902,0},{41902,0},{41902,0},{41902,0},{41902,0}]

Comment:
When reliability is 0, no message is received.

make run8: {task1, start, 400000, 100000} with reliability 50, process 3 terminate after 20000ms

<0.52.0>: [{100000,4808},
{100000,4829},
{100000,4853},

{100000,4814},
{100000,4786}]
<0.36.0>: [{100000,4846},
{100000,4862},
{100000,4830},
{100000,4792},
{100000,4717}]
<0.40.0>: [{100000,4761},
{100000,4833},
{100000,4810},
{100000,4785},
{100000,4798}]
<0.44.0>: [{20000,9104},{20000,8975},{20000,9117},{20000,9284},{20000,9105}]
<0.48.0>: [{100000,4738},
{100000,4759},
{100000,4800},
{100000,4772},
{100000,4812}]

Time used: 7 minutes 40 seconds
Comment: This is the most interesting make run we have, it runs 7 minutes 40 seconds, which is due to the nature of preemptive of Erlang, and, interestingly, the system3 actually receives more messages than others, and, it terminates after three systems that should actually terminate after itself.

Exercise 6 Conclusion

When we were investigating our system, it was first noticed that due to Erlang nature, all the processes are ran preemptive, and, the switch between process seems cheap. A process which should terminates in 1 minute actually terminate after more than 5 minute (includes a faulty process), which also proves that since send_after used in our system only records local time within a process.

The first feature we added was to make a process send and receive. Generally, we could see that system rapidly send messages by using after 0 in a receive clause. This design caused system inclined to send more message than receiving message, however, the APP is designed to receive rather than sending: the phenomenon has two reasons caused by communication between processes. One is, the message has a delay between sending and arriving, hence, the system will just rapidly sends before receive any. The second reason is that, the amounts of sent message is recorded in APP, but not any of RB, BEB, PL: APP updates the send map by adding one to all after it request a broadcast to all, however, the RB, BEB, PL shares a preemptive running environment with no priority/sleep strategy in APP. APP could have updates its map very quickly while sending and receiving are costly: the APP requested too much broadcasting and RB/BEB/PL failed to handle all of the message sending and receiving.

Another important feature we observed was, when RB, BEB, PL those additional layer was added, the performance of S/R message decreased, which should be expected, since APP now uses their agencies instead of communicating directly with each other.

Starting from system 4, simulation of lossy link is implemented, it has an interesting impact on S/R traffic. Conclusively, when link is lossy, PL will drop some of messages sending to pretend to be lossy, which will allow PL to have more chances to receive more, consequently, when more message is received, APP will be forced to check more message rather than send messages. Due to this phenomenon, we would like to say that message sending is more costly than other erlang commands.

In terms of forwarding strategy, it is implemented in system 6. In previous system, when MAX_MESSAGE is 0, all the APPs send 0 repeatedly, which is avoided by using random number in system 6. A signature of origin APP is included in message so that the repeating message is not forwarded, and the behavior of system is similar to the previous systems, except, its performance decreases, which is expected. The system provides ‘agreement’, so that if a correct process receives message M, then everyone else correct would receive M. We would like to say it is quite fascinating.

Finally, we would like to comment on faulty process 3: it acts normally until it dies, and in our system, other process would just keep sending message to process 3 rather than treating it as die. This is the best that could be done by BEB or RB.

Overall, we do think message sending is costly than usual commands in Erlang, and communication between process is asynchronized: the traffic bears more asynchronization since we are running all process in preemptive round robin style.