Master Ubinet
Distributed Algorithms
2012

F. Baude, L. Henrio

The marks for each exercise is only an indication.

1 Election (1.5pts)

We evaluated the Chang and Roberts election algorithm in the case all processes spontaneously participate initially in the election as candidates. In case processes are now forced to participate as candidates in the election, even in case they did'nt want to participate initially, what is the impact (if any) on the total number of exchanged messages and on the total time for having an elected leader. Does the ring organization (like the process numbers on the ring are stricly increasing, or, on the contrary are strictly decreasing) does matter when we evaluate the impact of this variant of the algorithm in term of complexity ?

2 Termination (synthesis question) (1pts)

Illustrate very shortly, taking some algorithms or applications[1] studied in the course as examples, why sometimes it is needed to run additionally a termination detection algorithm to be informed of termination of it; whereas in some cases, the detection of termination is intrinsically part of the algorithm that is being executed.

3 Group communications (7pts)

3.1 Properties of causal ordering

Compare the two following causal order properties:

1. If a process delivers messages m1 and m2 (to the application layer), and m1 $\rightarrow$ m2 then the process must deliver m1 before m2
2. If a process delivers a message m2, then it must have delivered every message m1 such that m1 $\rightarrow$ m2.

Are they equivalent ? If not, state which one is stronger. Which one is a safety property, and which one is a liveness property ? Which one should be used in the specification of a Causal Order Broadcast abstraction ?

3.2 About TOTAL and FIFO ordered multicast

Below is an algorithm, assuming no process faults can occur in the system, that ensures that each broadcasted message m is delivered in the same order on each process (ie TOTAL order is

[1]Think about the cryptarithmetic puzzle

ensured). For this, the algorithm relies upon a sequencer specific process. A process that whishes to TO-multicast a message m associates to it a supposely unique identifier, i. The algorithm relies upon a primitive B-multicast (B for basic), that ensures each process eventually gets the message. The group of processes to which a message is broadcasted contains the sequencer process

```
Algorithm for TO-Bcast on each process
--------------------------------------
On initialization, r:=0

To TO-Bcast message m:
B-multicast(<m,i>) // bcast the message to all, including to the sequencer process

On B-deliver(<m,i>):
Place <m,i> in hold-back queue

On B-deliver(<''order'',i,S>): {
wait until <m,i> in hold-back queue and S=r
// the wait does not prevent the process to react to another event, ie run an "On xxx :" code
TO-deliver(m)
r:=S+1 }

Algorithm for sequencer
-----------------------
On initialization s:=0
On B-deliver(<m,i>): {
B-multicast(<''order'',i,s>) // as order messages are not relevant for
//the sequencer process, we can consider that this B-multicast
//broadcasts this message "order" to all processes except the sequencer
s:=s+1 }
```

The B-multicast primitive used in this TO-multicast protocol supposes no guarantee regarding delivery ordering (in particular, if the same process Pi multicasts m1 , then m2, it could happen that TO-deliver protocol delivers m2 then m1).

1. Consequently, exhibit an example illustrating that it becomes possible that two causally ordered messages m1, m2, gets TO-delivered in the order m2, then m1.
2. Assuming the B-multicast primitive now ensures FIFO [2] ordering, show that this given TO-multicast algorithm becomes causally ordered.
3. Propose a protocol for the B-multicast so that it features FIFO ordering (we need such a protocol because we do not even assume that point to point communication links are FIFO!)
4. What is the complexity in number of exchanged messages for delivering a message m ? What is the paralell time complexity ?
5. How could you lower the total number of exchanged messages for broadcasting a message m ? Given your proposed improvement, is it still needed that you rely on a B-multicast that ensures FIFO ordering, and, in case not, what is the minimal assumption you need to put on end-to-end (process to process) communication links ?
6. (Bonus question) Even if you know only one example of a TOTAL order multicast algorithm, do you think that is it the case that any multicast that is both FIFO-ordered and totally ordered is thereby causally ordered ?

[2] If a (same) process issues B-multicast(m1) and then B-multicast(m2), then every process that B-delivers m1 will deliver m1 before m2

4 Fault-tolerance (7pts)

Here is a simple version of a CIC protocol algorithm we studied. Only message reception, message sending, and recovery are shown. `local_state` represents the whole local state of the process.

```
Receive message M from process i on process j with piggybacked checkpoint index N
 if N>current_checkpoint then
   send (local_state,j,N) to stable storage // forced checkpoint
   current_checkpoint = N
 endif
transmit message M to the application
END

Application sends message M from process i to process j
  Transmit message M to process j with piggybacked checkpoint index current_checkpoint
END

Recovery of global state N (executed by stable storage)
 for all processes i
  send to process i the state: (local_state,j,N)
 endfor
END
```

Questions (course application)

1. Suppose a proces crashes. How is N computed in the recovery function?
2. Write the pseudo-code of the function `Process i takes a local checkpoint`[3] (very short)
3. Explain in a few lines the advantage(s) of coordination induced checkpointing relatively to uncoordinated checkpoints

Problem The purpose of this problem is to adapt the CIC protocol to the case when one of the processes has special requirements, i.e. it is more difficult to control, and thus the other processes have to do whatever they can to adapt to this situation.

We suppose now that one of the processes in the system, the process P0, is not able to take a local checkpoint before the reception of a message. Thus the strict CIC protocol cannot be applied anymore. To overcome this issue, a first idea consists in *placing the forced checkpoint as soon as possible* after the message reception, that is just after the message transmission to the application. Concretely, a thread will take the forced checkpoint in parallel with the local execution. But, we suppose that there is a way to ensure that the forced checkpoint occurs just after the message reception (not concurrently with the rest of the process execution whose role is to effectively process the transmitted message).

1. Write the new pseudo-code for `Receive message` (for the process P0 only) – we suppose that "Transmit message" delegates a thread for the handling of the message and the checkpoint can be taken just after message reception.
2. On Figure A (last page) draw the forced checkpoints, the checkpoint numbers, and the piggybacked numbers.
3. Consider lines formed with identical checkpoint numbers; when do those lines form consistent recovery lines? Why? Consequently, can you use the checkpoint numbers to compute a recovery line?

[3]needed in CIC, even if not explicitly called in the above algorithm

4. What is the best recovery line if process P0 fail? if process P1? **To compute/check the best recovery line,draw the rollback-dependency graph on the figure.**
5. Is the protocol efficient for preventing messages emitted by P0 to be orphan? and messages received by P0? why?
6. conclude on the usefulness/efficiency of this protocol

A second strategy consists in taking a checkpoint *before sending* each message from P0 **and** *before seding a message to P0*

1. Write the new pseudo-code for `send message` (the receive message function is the unmodified one shown at the begining of this exercise)
2. On Figure B (last page) draw the forced checkpoints, the checkpoint numbers, and the piggybacked numbers.
3. Consider lines formed with identical checkpoint numbers; do those lines form consistent recovery lines? Why? Consequently, can you use the checkpoint numbers to compute a recovery line?
4. what is the best recovery line if process P0 fail? if process P1? **To compute/check the best recovery line,draw the rollback-dependency graph on the figure.**
5. Is the protocol efficient for preventing messages emitted by P0 to be orphan? and messages received by P0? why?
6. conclude on the usefulness/efficiency of this protocol

Bonus question (difficult): improve this second protocol to prevent the existence (or at least reduce the number) of orphan messages.

5 Mutual exclusion problem (3pts)

We want to provide a mutual exclusion protocol for a system where the process P0 is considered as prioritary and should be able to enter its critical section as soon as possible, possibly overtaking the other processes.

Consider the token ring approach, which grounds upon the transmission of a token:

1. How would you modify the token ring protocol so that the process P0 can enter its CS as soon as possible, i.e. waiting at maximum that one other process does its critical section?
2. Write the algorithm executed in P0; in each process Pi with $i \neq 0$
3. Consider a process other than P0, how long does it have to wait before entering its critical section (in the worst case, and counting in number of critical sections executed)

Same questions for the Suzuki-Kasami algorithm (it is one of those you studied as homework - the one which use queues as tokens)

1. How would you modify the Suzuki-Kasami protocol so that the process P0 can enter its CS as soon as possible, i.e. waiting at maximum that one other process does its critical section?
2. Write the algorithm executed in P0; in each process Pi with $i \neq 0$
3. Consider a process other than P0, how long does it have to wait before entering its critical section (in the worst case, and counting in number of critical sections executed)

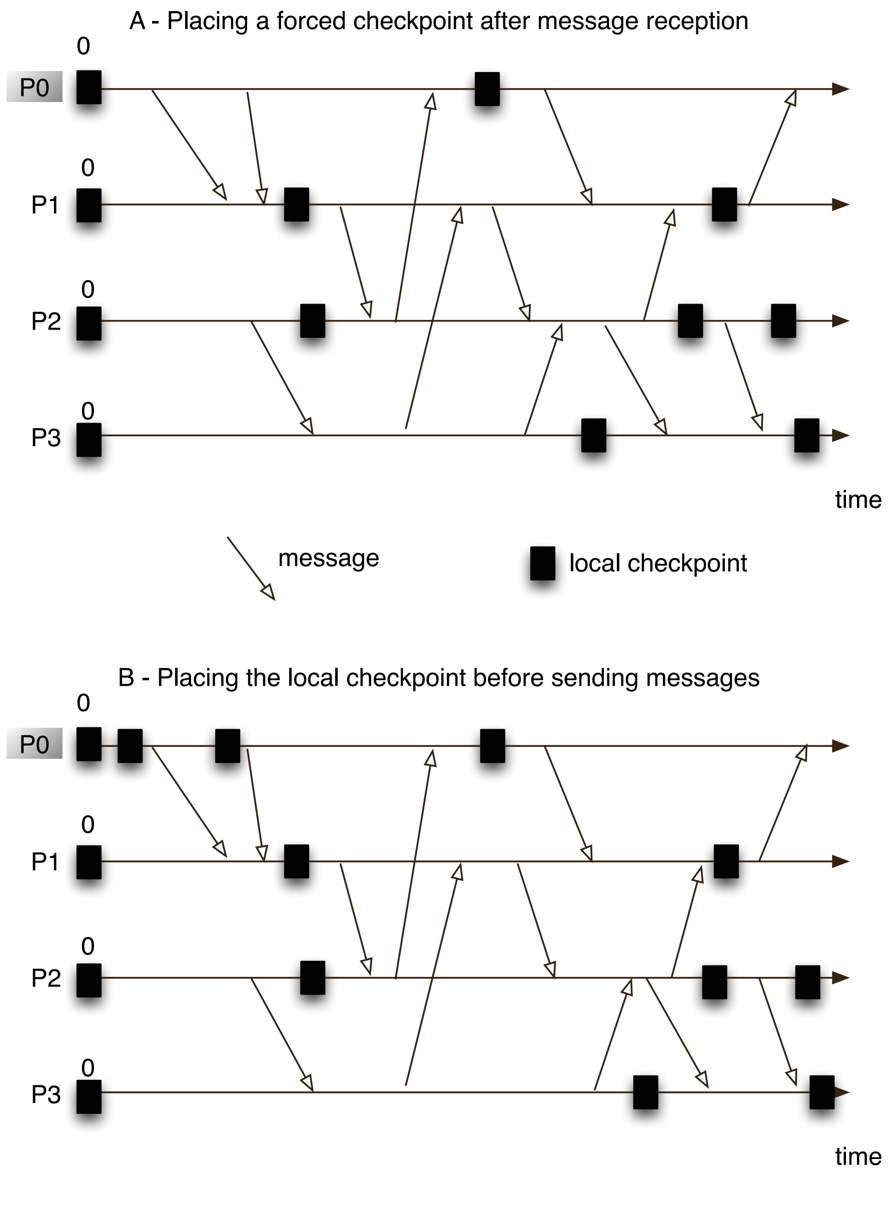
A - Placing a forced checkpoint after message reception
0
P0
0
P1
0
P2
0
P3
time
message
local checkpoint
B - Placing the local checkpoint before sending messages
0
P0
0
P1
0
P2
0
P3
time

Figure 1: An execution with some messages: **Fault-tolerance problem**