4.7 ALGORITHM FOR ASYNCHRONOUS CHECKPOINTING AND RECOVERY:

The algorithm of Juang and Venkatesan for recovery in a system that uses asynchronous check pointing.

A. System Model and Assumptions

The algorithm makes the following assumptions about the underlying system:

- The communication channels are reliable, deliver the messages in FIFO order and have infinite buffers.
- The message transmission delay is arbitrary, but finite.
- Underlying computation/application is event-driven: process P is at state s, receives message m, processes the message, moves to state s' and send messages out. So the triplet (*s, m, msgs_sent*) represents the state of P

Two type of log storage are maintained:

- Volatile log: short time to access but lost if processor crash. Move to stable log periodically.
- Stable log: longer time to access but remained if crashed

A. Asynchronous Check pointing

- After executing an event, the triplet is recorded without any synchronization with other processes.
- Local checkpoint consist of set of records, first are stored in volatile log, then moved to stable log.

B. The Recovery Algorithm

Notations and data structure

The following notations and data structure are used by the algorithm:

• RCVDi←j(CkPti) represents the number of messages received by processor pi from processor pj , from the beginning of the computation till the checkpoint CkPti.

• SENTi→j(CkPti) represents the number of messages sent by processor pi to processor pj , from the beginning of the computation till the checkpoint CkPti.

Basic idea

- Since the algorithm is based on asynchronous check pointing, the main issue in the recovery is to find a consistent set of checkpoints to which the system can be restored.
- The recovery algorithm achieves this by making each processor keep track of both the

number of messages it has sent to other processors as well as the number of messages it has received from other processors.

- Whenever a processor rolls back, it is necessary for all other processors to find out if any message has become an orphan message. Orphan messages are discovered by comparing the number of messages sent to and received from neighboring processors.

For example, if RCVDi←j(CkPti) > SENTj→i(CkPtj) (that is, the number of messages received by processor pi from processor pj is greater than the number of messages sent by processor pj to processor pi, according to the current states the processors), then one or more messages at processor pj are orphan messages.

The Algorithm

When a processor restarts after a failure, it broadcasts a ROLLBACK message that it had failed

Procedure RollBack_Recovery

processor pi executes the following:

STEP (a)

if processor pi is recovering after a failure **then**

CkPti := latest event logged in the stable storage

else

CkPti := latest event that took place in pi {The latest event at pi can be either in stable or in volatile storage.}

end if

STEP (b)

for k = 1 1 to N {N is the number of processors in the system} **do**

for each neighboring processor pj **do**

compute SENTi→j(CkPti)

send a ROLLBACK(i, SENTi→j(CkPti)) message to pj

end for

for every ROLLBACK(j, c) message received from a neighbor j **do**

if RCVDi←j(CkPti) > c {Implies the presence of orphan messages} **then**

find the latest event e such that RCVDi←j(e) = c {Such an event e may be in the volatile storage or stable storage.}

CkPti := e

end if

end for

end for{for k}

D. An Example

Consider an example shown in Figure 2 consisting of three processors. Suppose processor Y fails and restarts. If event ey2 is the latest checkpointed event at Y, then Y will restart from the state corresponding to ey2.

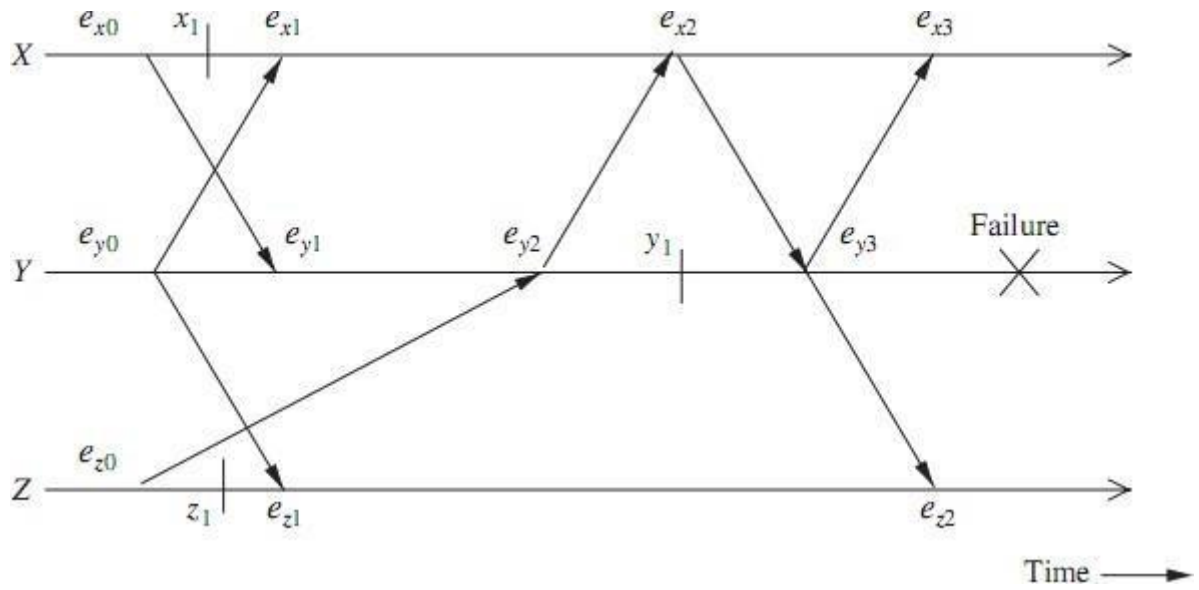

Figure 2: An example of Juan-Venkatesan algorithm.

- Because of the broadcast nature of ROLLBACK messages, the recovery algorithm is initiated at processors X and Z.
- Initially, X, Y, and Z set CkPtX ← ex3, CkPtY ← ey2 and CkPtZ ← ez2, respectively, and X, Y, and Z send the following messages during the first iteration:
- Y sends ROLLBACK(Y,2) to X and ROLLBACK(Y,1) to Z;
- X sends ROLLBACK(X,2) to Y and ROLLBACK(X,0) to Z;
- Z sends ROLLBACK(Z,0) to X and ROLLBACK(Z,1) to Y.

Since RCVDX←Y (CkPtX) = 3 > 2 (2 is the value received in the ROLLBACK(Y,2) message from Y), X will set CkPtX to ex2 satisfying RCVDX←Y (ex2) = 1≤ 2.

Since RCVDZ←Y (CkPtZ) = 2 > 1, Z will set CkPtZ to ez1 satisfying RCVDZ←Y (ez1) = 1 ≤ 1.

At Y, RCVDY←X(CkPtY) = 1 < 2 and RCVDY←Z(CkPtY) = 1 = SENTZ←Y (CkPtZ).

Y need not roll back further.

In the second iteration, Y sends ROLLBACK(Y,2) to X and ROLLBACK(Y,1) to Z;

Z sends ROLLBACK(Z,1) to Y and ROLLBACK(Z,0) to X;

X sends ROLLBACK(X,0) to Z and ROLLBACK(X, 1) to Y.

If Y rolls back beyond ey3 and loses the message from X that caused ey3, X can resend this message to Y because ex2 is logged at X and this message available in the log. The second and third iteration will progress in the same manner. The set of recovery points chosen at the end of the first iteration, {ex2, ey2, ez1}, is consistent, and no further rollback occurs.