COORDINATED CHECKPOINTING ALGORITHM (KOO-TOUEG)

- Koo and Toueg coordinated check pointing and recovery technique takes a consistent set of checkpoints and avoids the domino effect and live lock problems during the recovery.
- Includes 2 parts: the check pointing algorithm and the recovery algorithm

A. The Check pointing Algorithm

The checkpoint algorithm makes the following assumptions about the distributed system:

- Processes communicate by exchanging messages through communication channels.
- Communication channels are FIFO.
- Assume that end-to-end protocols (the sliding window protocol) exist to handle with message loss due to rollback recovery and communication failure.
- Communication failures do not divide the network.
- The checkpoint algorithm takes two kinds of checkpoints on the stable storage: Permanent and Tentative.
- A *permanent checkpoint* is a local checkpoint at a process and is a part of a consistent global checkpoint.
- A *tentative checkpoint* is a temporary checkpoint that is made a permanent checkpoint on the successful termination of the checkpoint algorithm.

The algorithm consists of two phases.

First Phase

1. An initiating process Pi takes a tentative checkpoint and requests all other processes to take tentative checkpoints. Each process informs Pi whether it succeeded in taking a tentative checkpoint.
2. A process says "no" to a request if it fails to take a tentative checkpoint
3. If Pi learns that all the processes have successfully taken tentative checkpoints, Pi decides that all tentative checkpoints should be made permanent; otherwise, Pi decides that all the tentative checkpoints should be thrown-away.

Second Phase

1. Pi informs all the processes of the decision it reached at the end of the first phase.
2. A process, on receiving the message from Pi will act accordingly.

3. Either all or none of the processes advance the checkpoint by taking permanent checkpoints.
4. The algorithm requires that after a process has taken a tentative checkpoint, it cannot send messages related to the basic computation until it is informed of Pi's decision.

Correctness: for two reasons

i. Either all or none of the processes take permanent checkpoint
ii. No process sends message after taking permanent checkpoint

An Optimization

The above protocol may cause a process to take a checkpoint even when it is not necessary for consistency. Since taking a checkpoint is an expensive operation, we avoid taking checkpoints.

B. The Rollback Recovery Algorithm

The rollback recovery algorithm restores the system state to a consistent state after a failure. The rollback recovery algorithm assumes that a single process invokes the algorithm. It assumes that the checkpoint and the rollback recovery algorithms are not invoked concurrently. The rollback recovery algorithm has two phases.

First Phase

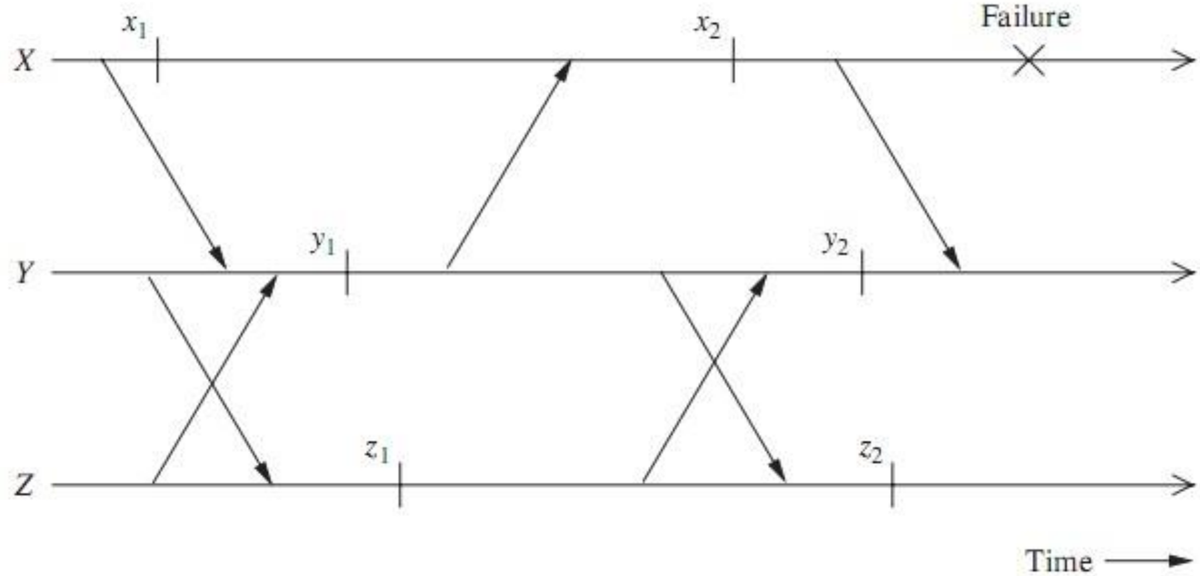
1. An initiating process Pi sends a message to all other processes to check if they all are willing to restart from their previous checkpoints.
2. A process may reply "no" to a restart request due to any reason (e.g., it is already participating in a check pointing or a recovery process initiated by some other process).
3. If Pi learns that all processes are willing to restart from their previous checkpoints, Pi decides that all processes should roll back to their previous checkpoints. Otherwise,
4. Pi aborts the roll back attempt and it may attempt a recovery at a later time.

Second Phase

1. Pi propagates its decision to all the processes.
2. On receiving Pi's decision, a process acts accordingly.
3. During the execution of the recovery algorithm, a process cannot send messages related to the underlying computation while it is waiting for Pi's decision.

Correctness: Resume from a consistent state

Optimization: May not to recover all, since some of the processes did not change anything



The above protocol, in the event of failure of process X, the above protocol will require processes X, Y, and Z to restart from checkpoints x2, y2, and z2, respectively. Process Z need not roll back because there has been no interaction between process Z and the other two processes since the last checkpoint at Z.

ALGORITHM FOR ASYNCHRONOUS CHECKPOINTING AND RECOVERY (JUANG-VENKATESAN)

- This algorithm helps in recovery in asynchronous checkpointing.
- The following are the assumptions made:
  - communication channels are reliable
  - delivery messages in FIFO order
  - infinite buffers
  - message transmission delay is arbitrary but finite
- The underlying computation or application is event-driven: When process P is at states, receives message m, it processes the message; moves to state s' and send messages out. So the triplet (s, m, msgs_sent) represents the state of P.
- To facilitate recovery after a process failure and restore the system to a consistent state, two types of log storage are maintained:
  - **Volatile log:** It takes short time to access but lost if processor crash. Thecontents of volatile log are moved to stable log periodically.
  - **Stable log:** longer time to access but remained if crashed.

Asynchronous checkpointing

- After executing an event, a processor records a triplet (s, m, msg_sent) in its volatile storage.
  - s:state of the processor before the event
  - m: message
  - msgs_sent: set of messages that were sent by the processor during the event.
- A local checkpoint at a processor consists of the record of an event occurring at the processor and it is taken without any synchronization with other processors.
- Periodically, a processor independently saves the contents of the volatile log in the stable storage and clears the volatile log.
- This operation is equivalent to taking a local checkpoint.

Recovery Algorithm

The data structures followed in the algorithm are:

$RCVD_{i\to j}(CkPt_i)$ This represents the number of messages received by processor pi from processor pj, from the beginning of the computation until the checkpoint $CkPt_i$.

$$SENT_{i\to j}(CkPt_i)$$

This represents the number of messages sent by processor pi to processor pj, from the beginning of the computation until the checkpoint $CkPt_i$.

- The main idea of the algorithm is to find a set of consistent checkpoints, from theset of checkpoints.
- This is done based on the number of messages sent and received.
- Recovery may involve multiple iterations of roll backs by processors.
- Whenever a processor rolls back, it is necessary for all other processors to find outif any message sent by the rolled back processor has become an orphan message.
- The orphan messages are identified by comparing the number of messages sent to and received from neighboring processors.
- When a processor restarts after a failure, it broadcasts a ROLLBACK message thatit has failed.
- The recovery algorithm at a processor is initiated when it restarts after a failure or when it learns of a failure at another processor.
- Because of the broadcast of ROLLBACK messages, the recovery algorithm is initiated at all processors.

Procedure RollBack_Recovery: processor pi executes the following:STEP (a)

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

$C_k Pt_i$:= latest event logged in the stable storage

else

$C_k Pt_i$:= latest event that look place in p_i {The latest event at pi can be either instable or in volatile storage}

end if

STEP(b)

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

for each neighboring processor p_j **do**

compute $SENT_{i\to j}(C_k Pt_i)$

send a ROLLBACK(i, $SENT_{i\to j}(C_k Pt_i)$) message to p_j

end for

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

if RCVD $_{i\rightarrow j}$ (C_k Pt_i) > c {Implies the presence of orphan message}

then

find the latest event e such that RCVD $_{i\rightarrow j}$ (e) = c {Such an event e may be in the volatile storage or stable storage}

C_k Pt_i := e

end if

end for

end for {for k}

Fig : Algorithm for Asynchronous Check pointing and Recovery (Juang- Venkatesan)

- The rollback starts at the failed processor and slowly diffuses into the entire systemthrough ROLLBACK messages.
- During the kth iteration (k != 1), a processor pi does the following:

(i) based on the state $CkPt_i$ it was rolled back in the (k − 1)th iteration, it computes $SENT_{i\rightarrow j}$ ($CkPt_i$) for each neighbor pj and sends this value in a ROLLBACK message to that neighbor

(ii) p_i waits for and processes ROLLBACK messages that it receives from its neighbors in kth iteration and determines a new recovery point $CkPt_i$ for p_i based on information in these messages.

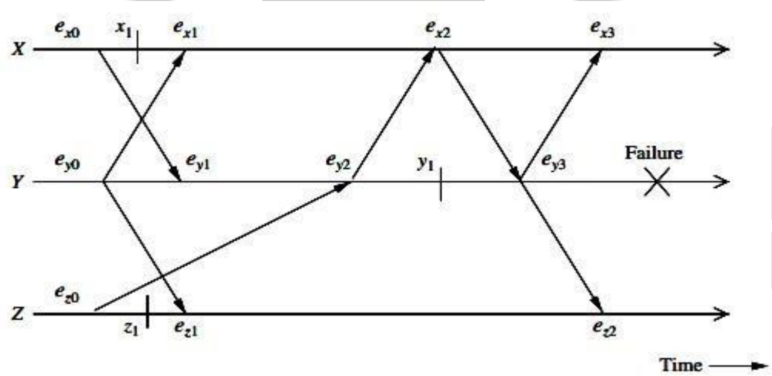


Fig : Asynchronous Checkpointing And Recovery

At the end of each iteration, at least one processor will rollback to its final recovery point, unless the current recovery points are already consistent.