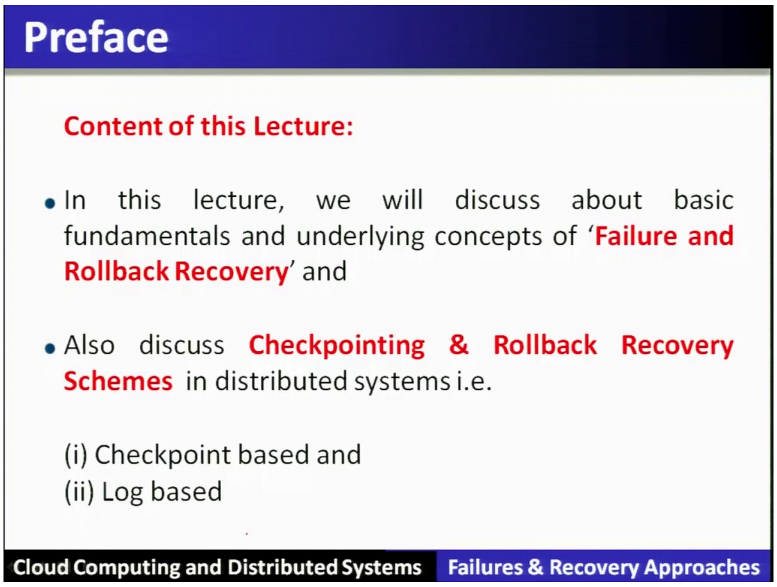
# Cloud Computing and Distributed Systems

**Dr. Rajiv Misra**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Patna**

**Lecture – 15**
**Failures and Recovery Approaches in Distributed Systems**

(Refer Slide Time: 00:19)



## Preface

**Content of this Lecture:**

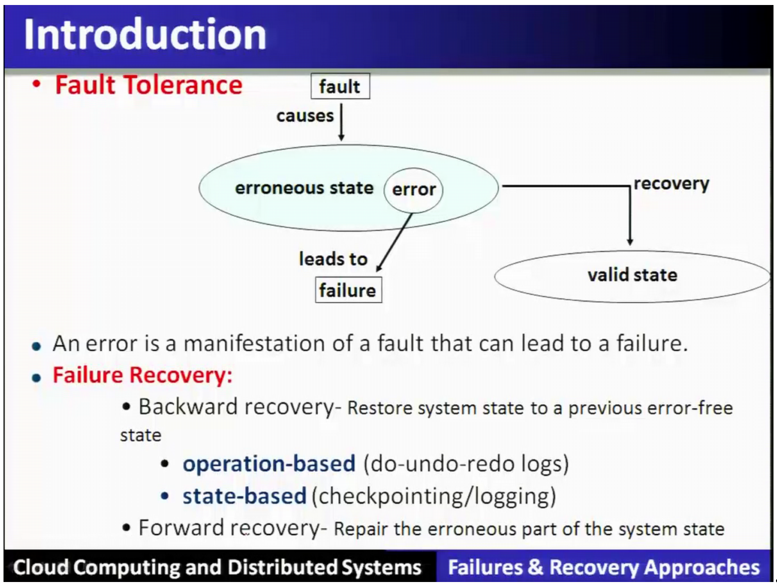
- In this lecture, we will discuss about basic fundamentals and underlying concepts of '**Failure and Rollback Recovery**' and
- Also discuss **Checkpointing & Rollback Recovery Schemes** in distributed systems i.e.

(i) Checkpoint based and
(ii) Log based

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Failure and Recovery Approaches in Distributed Systems. Preface content of this lecture, we will discuss about basic fundamentals and underlying concepts of Failure and Rollback Recovery in distributed systems. Also we will discuss the Checkpointing and Rollback Recovery Schemes of distribute systems that is, checkpointing based schemes and log based schemes.

(Refer Slide Time: 00:44)



## Introduction

- **Fault Tolerance**

fault
causes
erroneous state
error
recovery
valid state
leads to
failure

- An error is a manifestation of a fault that can lead to a failure.
- **Failure Recovery:**
  - Backward recovery- Restore system state to a previous error-free state
    - **operation-based** (do-undo-redo logs)
    - **state-based** (checkpointing/logging)
  - Forward recovery- Repair the erroneous part of the system state

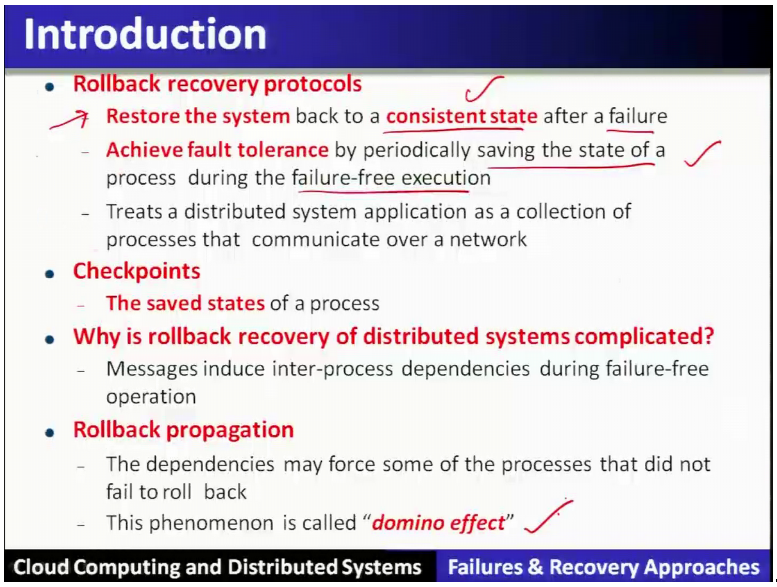
Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Introduction; so, error is a manifestation of a fault, that will lead to a failure. Here we will see that, in that particular state which is called erroneous state. The recovery is required to reach to a valid state which is an error free state.

So, the fault tolerance is achieved using recovery of a failure using two methods. The first one is called forward recovery method where the repair of the erroneous part of a system state is known and it will be carried out to reach or to recover the system from failed state to the valid state that is called forward recovery method. There is another failure recovery method which is called, a backward recovery method. It will restore the system state to a previous error free state; it does not require the knowledge of the erroneous part of the system state.

The backward recovery is of two types, that is operation based backward recovery method in which all the course grained operation like do, undo, redo. They are logged in a file and they are being operated on, that is why it is called operation based. The second type of backward recovery is called a state based. Here the states are stored or a logged, they are also called checkpointing. So, when a failure happens using this particular checkpointing or a log based logging method, it will try to recover to a restore the previous valid state.

(Refer Slide Time: 02:56)



## Introduction

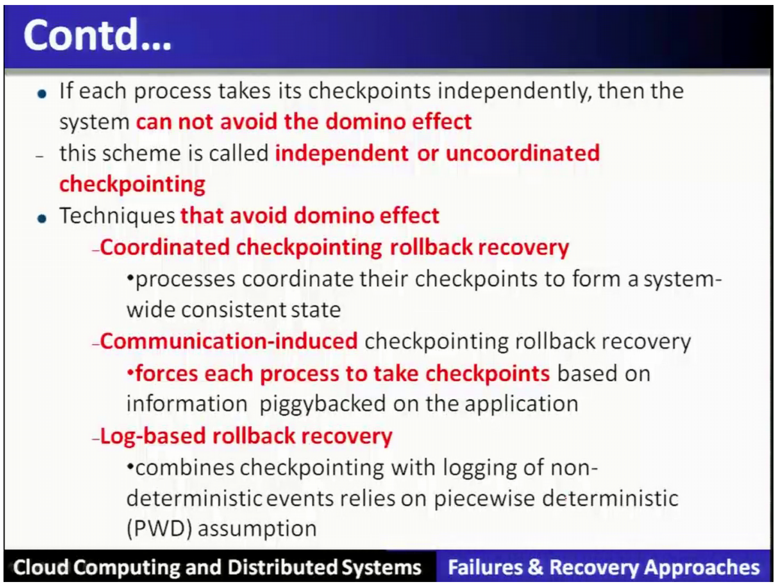
- Rollback recovery protocols
  - Restore the system back to a consistent state after a failure
  - Achieve fault tolerance by periodically saving the state of a process during the failure-free execution
  - Treats a distributed system application as a collection of processes that communicate over a network
- Checkpoints
  - The saved states of a process
- Why is rollback recovery of distributed systems complicated?
  - Messages induce inter-process dependencies during failure-free operation
- Rollback propagation
  - The dependencies may force some of the processes that did not fail to roll back
  - This phenomenon is called "*domino effect*"

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

So, the rollback recovery protocols will restore the system from the previous consistent state after the failure and it achieves the fault tolerance by periodically saving the state of a process during the failure free execution. That means, to restore the system from the previous error free state that is called consistent state. After the failure, it requires to maintain while the system is failure free that is periodically saving the states which are called consistent state. So, that during the failure it can restart from that point onwards. This is going to be an important issue, how to save the state, so that the recovery or the system can recover whenever there is a fault happens. Now, here we treat the distributed system application as a collection of processes that they communicate over the network. In that model how the recording of the states are done, when the while in the failure free executions.

So, one such method is called a checkpoint; that means, checkpoint is a saved state of a process. So, why is the rollback recovery, in a distributed system is complicated. So, messages induced by inter process dependencies during the failure free operation and capturing this, particular dependencies which will be useful for the recovery is a complicated. One: Rollback propagation, these dependencies may force some of the processes which do not fail to rollback and this phenomenon is called domino effect.

(Refer Slide Time: 05:20)



## Contd...

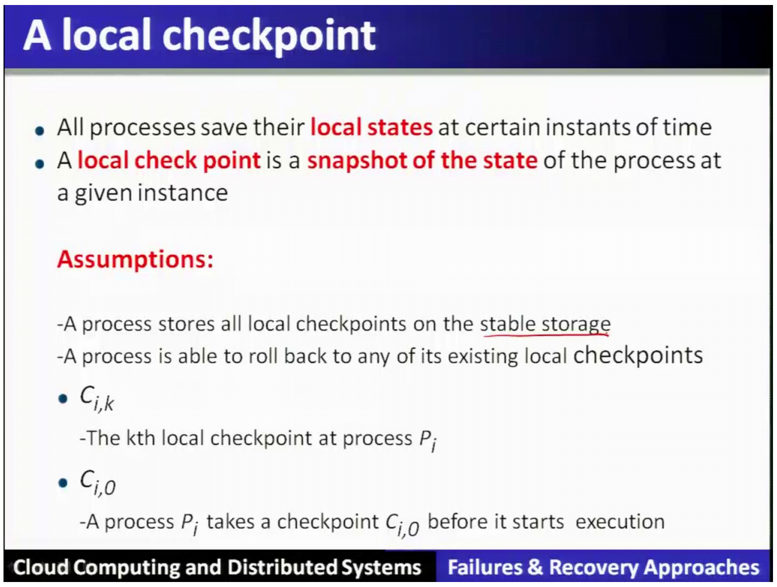
- If each process takes its checkpoints independently, then the system **can not avoid the domino effect**
- this scheme is called **independent or uncoordinated checkpointing**
- Techniques **that avoid domino effect**
  - **Coordinated checkpointing rollback recovery**
    - •processes coordinate their checkpoints to form a system-wide consistent state
  - **Communication-induced** checkpointing rollback recovery
    - **•forces each process to take checkpoints** based on information piggybacked on the application
  - **Log-based rollback recovery**
    - •combines checkpointing with logging of non-deterministic events relies on piecewise deterministic (PWD) assumption

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

If each process takes its checkpoint independently, then the system cannot avoid domino effect. This scheme is called independent or uncoordinated checkpointing. The advantage of uncoordinated checkpointing is that, checkpointing can be done easily without much overhead, but the drawback is that it suffers from domino effect. So, the techniques that can avoid domino effect are coordinated checkpointing and rollback recovery method; that means the processes. They coordinate to record their checkpoints to form the system wide consistent state. Therefore, with the coordination or with the coordinated checkpointing, this state that is called domino effect will not happen and all the state recorded is a consistent state.

Communication induced checkpointing rollback recovery method forces each process to take checkpoints based on the information which is piggybacked on the application, also useful to avoid the domino effect while taking the checkpoints. Third scheme is called log based rollback recovery scheme. This combines the checkpointing with the logging of non deterministic events which relies on a piecewise deterministic assumptions. So that means, log based rollback is nothing, but the combined checkpointing with the logging of non deterministic events; that means, that dependencies are properly exploited wire while taking the checkpoints. So, that the consistent state is being recorded. So, in all three cases we see that, we can avoid the checkpointing, but it requires some effort at the time of checkpointing.

(Refer Slide Time: 07:21)



## A local checkpoint

- All processes save their **local states** at certain instants of time
- A **local check point** is a **snapshot of the state** of the process at a given instance

**Assumptions:**

-A process stores all local checkpoints on the stable storage
-A process is able to roll back to any of its existing local checkpoints

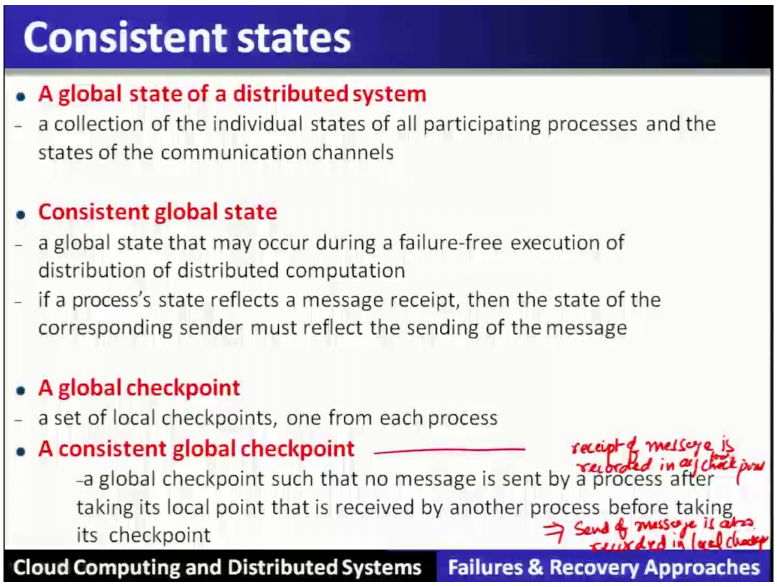
- $C_{i,k}$
  -The kth local checkpoint at process $P_i$
- $C_{i,0}$
  -A process $P_i$ takes a checkpoint $C_{i,0}$ before it starts execution

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Let us see the preliminaries. A local checkpoint that is, all the processes save their local state at a certain instant of time is, so a local checkpoint is a snapshot of the state of a process at a given instance. So, let us assume some of the notations. A process stores all the local checkpoints on the stable storage. So, stable storage is that kind of storage which will survive even after the serious faults.

So, a process is able to rollback to any of the previous existing local checkpoints. Let C i, k denote the kth local checkpoint at a process i and C i,0 denotes the process i, which takes the checkpoints C i, 0, before it is starts the execution.

(Refer Slide Time: 08:23)



## Consistent states

- **A global state of a distributed system**
  - a collection of the individual states of all participating processes and the states of the communication channels
- **Consistent global state**
  - a global state that may occur during a failure-free execution of distribution of distributed computation
  - if a process's state reflects a message receipt, then the state of the corresponding sender must reflect the sending of the message
- **A global checkpoint**
  - a set of local checkpoints, one from each process
- **A consistent global checkpoint** — receipt of message is recorded in any local checkpoint
  - a global checkpoint such that no message is sent by a process after taking its local point that is received by another process before taking its checkpoint

$\Rightarrow$ Send of message is also recorded in local checkp

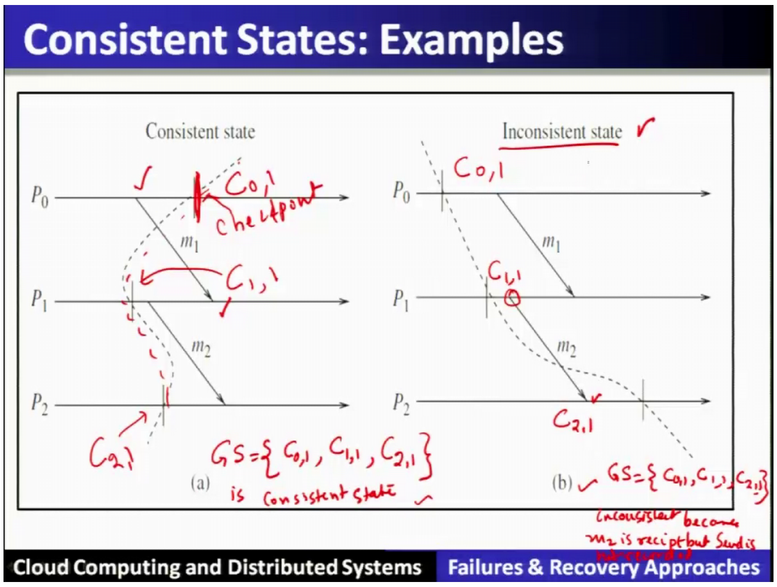
Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Now, let us see some of the definitions. A global state of a distributed system is a collection of individual states of all participating processes and the state of a communication channel that, particular global state which is called a consistent global state may occur during the failure free execution of a distributed computation.

Now, if a process state reflects a message receipt, then the state of a corresponding sender of that particular message must also reflect the sending of a message. If, this condition is captured in the consistent global state or if this condition is captured in a recording of a global state, then that global state is called a consistent global state. What is a global checkpoint? A set of local checkpoint, from each process is called a global checkpoint that, global checkpoint which holds the following condition is called a global checkpoint.

So, that conditions says, that a global checkpoint such that, no message is sent by a process after taking its local checkpoint that is received by another process, before taking its checkpoint; meaning to say that in a consistent global checkpoint, if the receipt of a message is recorded in any checkpoint in any local checkpoint; then this implies that the send of the message is also recorded in the local checkpoint. Hence this is called a global checkpoint, we will see these details.

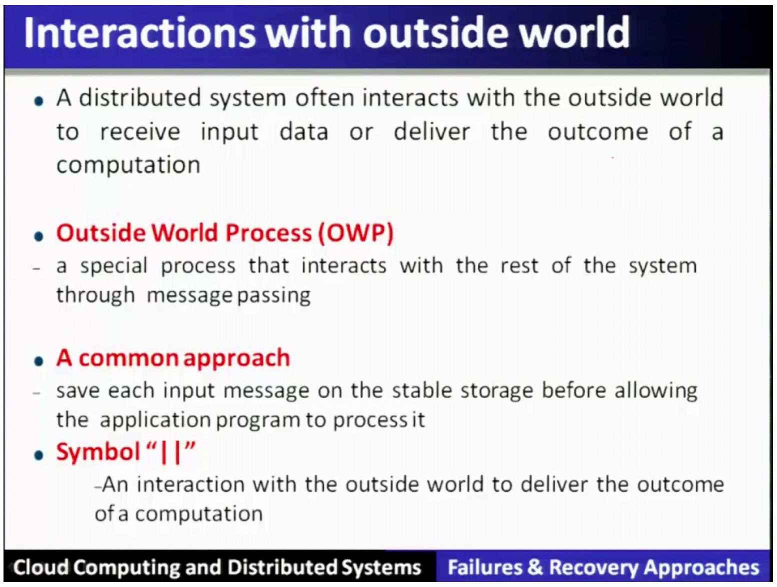
(Refer Slide Time: 10:41)


Consistent States: Examples
Consistent state
Inconsistent state
$P_0$
$P_1$
$P_2$
$m_1$
$m_2$
$C_{0,1}$
checkpoint
$C_{1,1}$
$C_{2,1}$
$GS=\{C_{0,1}, C_{1,1}, C_{2,1}\}$ is consistent state
(a)
(b)
inconsistent because m2 is receipt but send is
Cloud Computing and Distributed Systems
Failures & Recovery Approaches

Here, we can we have shown the consistent state with a dotted line. So, that means, this vertical bar shows the checkpoint, let us say this is C 0 1 and this is C 1 1 and here this is called C 2 1. This particular state that, is the global state which is defined by the checkpoints; C 0 comma 1, C 1 comma 1, C 2 comma 1. This particular global state is a consistent state. Because, let us see this particular message, the send is recorded but receive is not recorded that is possible in the set of global state or in the set of checkpoints.

Therefore, this is a consistent state however; on the other hand this example the other example, which is shown in the figure b is inconsistent state. Let us see the global state, this is C 0 comma 1, this is C 1 comma 1, this is C 2 comma 1, so that collection C 0 comma 1, C 1 comma 1, C 2 comma 1. This particular global state is inconsistent because, there is a receipt of a message m 2 but its sent is not recorded. Hence this is going to be an inconsistent global state

(Refer Slide Time: 13:22)



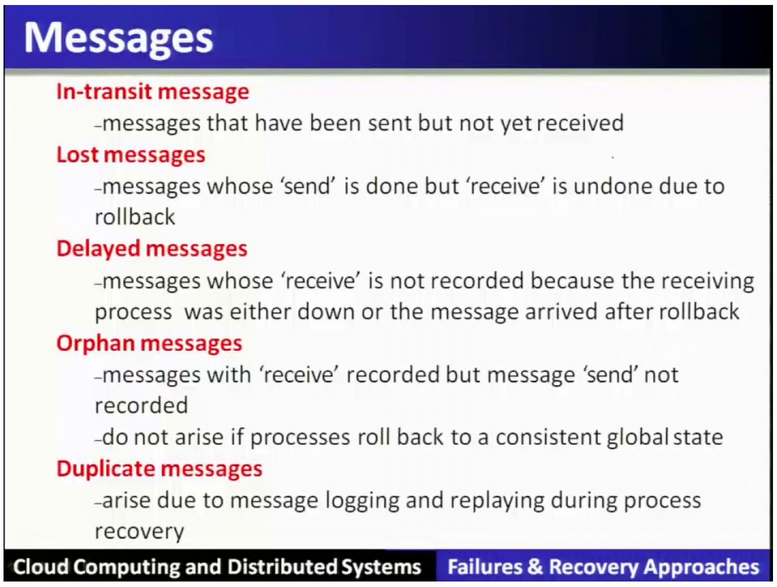
## Interactions with outside world

- A distributed system often interacts with the outside world to receive input data or deliver the outcome of a computation
- **Outside World Process (OWP)**
  - a special process that interacts with the rest of the system through message passing
- **A common approach**
  - save each input message on the stable storage before allowing the application program to process it
- **Symbol "| |"**
  - An interaction with the outside world to deliver the outcome of a computation

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Interaction with the outside world, the distributed system of interacts with the outside world to receive the input data or deliver the outcome of the computation, which is denoted as outside world process. This is a special process that interacts with the rest of the world through the message passing. A common approach is to, save each input message on the storage before allowing the application program to process it. So, the interaction with the outside world, to deliver the outcome of a computation is denoted by the symbol double bar.

(Refer Slide Time: 14:05)



## Messages

**In-transit message**
- messages that have been sent but not yet received

**Lost messages**
- messages whose 'send' is done but 'receive' is undone due to rollback

**Delayed messages**
- messages whose 'receive' is not recorded because the receiving process was either down or the message arrived after rollback

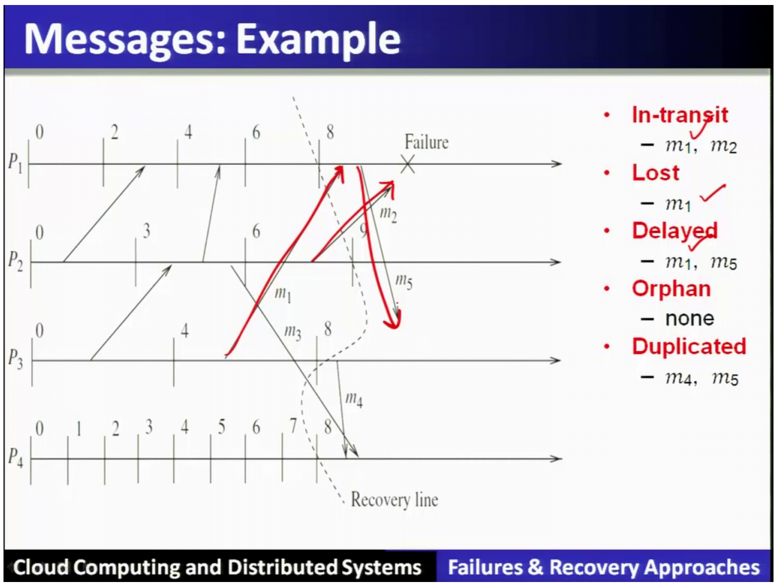
**Orphan messages**
- messages with 'receive' recorded but message 'send' not recorded
- do not arise if processes roll back to a consistent global state

**Duplicate messages**
- arise due to message logging and replaying during process recovery

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

We will see different type of messages, which are used in our system for recording, as well as to deal with the failure recovery. So, let us understand them and then, we will see how to deal with them, which are serious and which are not that serious, to be for the handling during the recovery method So, the messages is which are called in transit messages, they have been sent but not yet received, that is called in transit message.

(Refer Slide Time: 14:48)



## Messages: Example

- In-transit
  - $m_1$, $m_2$
- Lost
  - $m_1$
- Delayed
  - $m_1$, $m_5$
- Orphan
  - none
- Duplicated
  - $m_4$, $m_5$

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

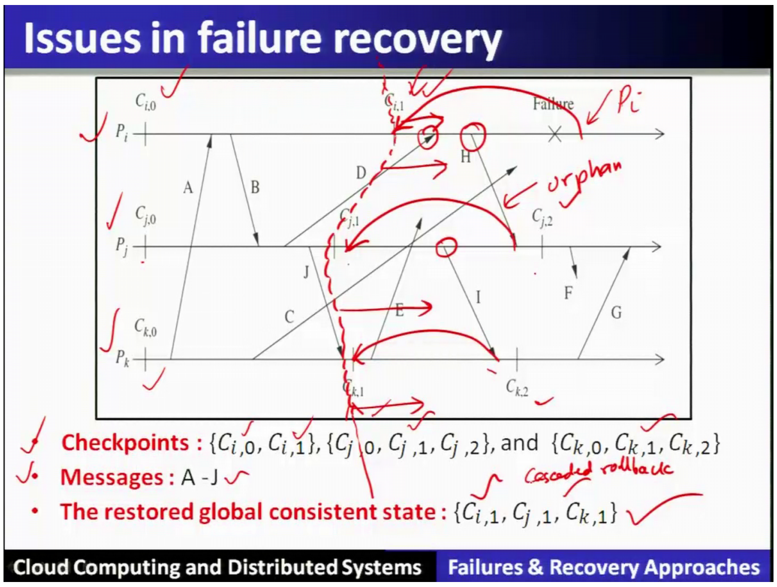
So, here in this particular figure the message m 1 and m 2, so, m 2 is this particular message this is sent, but not being delivered.

Similarly, the message m 1 is also in the transit why because, it is sent but the receiving process that is p 1 is either not available due to the failure. Hence this message will be in the transit, will be considered in the transit. Lost messages, that is messages who send is done, but receive is undone due to the rollback. Here the last message is m 1 why because; the failure will not allow the p 1 to receive the message. Hence this particular message is also transit and also in the lost message.

Delayed messages, that is messages whose receive is not recorded because the receiving process was either down or the message arrived late after the rollback. Here the again the delayed message m 1 is also denoted as the delayed message and another message m 5 is also, this m 5 is also delayed. Orphan message is, so messages with receive recorded, but its send of that particular messages in the checkpointing this is called an orphan message. So, let us see, here it is not shown, we will discuss this kind of message later on.

Duplicate messages arise due to the message logging and replying during the process recovery.

(Refer Slide Time: 16:41)



## Issues in failure recovery

- Checkpoints : $\{C_{i,0}, C_{i,1}\}, \{C_{j,0}, C_{j,1}, C_{j,2}\}$, and $\{C_{k,0}, C_{k,1}, C_{k,2}\}$
- Messages : A - J
- The restored global consistent state : $\{C_{i,1}, C_{j,1}, C_{k,1}\}$

Cascaded rollback

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

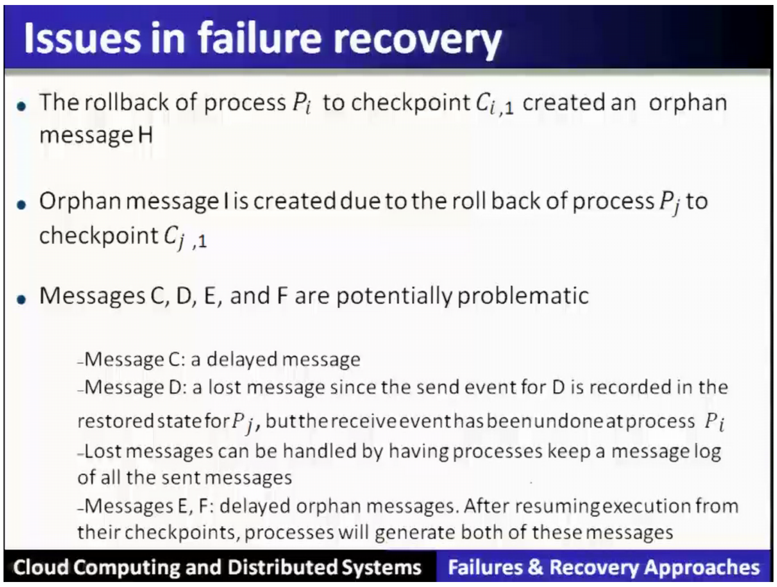
Now, here let us understand the figure and then we will see these issues. So, the messages which are shown here, from A to J they are being labeled. Also the checkpoints the process P 1 P i has taken the 2 checkpoints they are C i 0 and C i 1 similarly process P J has taken 1, 2 and 3 checkpoints, which are mentioned over here. Similarly P K has taken 3 checkpoints 1, see, 1, 2 and 3. Now if there is a failure at this point of time in shown over here there is a failure then, the recovery requires the system to restart from the global consistent state that is, the most recent global consistent state. Let us assume that, P 1 will restart from this checkpoint that is C i comma 1 then, this particular receive of this particular message which if, it is already happened then it will not be received.

Similarly, the send of a message H, is also not being considered now in this scenario. So, if send is not recorded, but if this checkpoint C J 2 is being considered, it has recorded the received, but send due to the failure is not recorded. So this becomes an orphan message which, in turn will trigger the rollback to the previous consistent state. So, this will trigger the rollback over here, this will trigger the roll back to the previous consistent state. Now this particular rollback will say that this particular message send, it is send is not recorded now, therefore, this particular message which is received it is send is not recorded. Therefore, the system this will trigger to rollback from this position.

Therefore, this will be the consistent checkpoint and the system will restart from this particular state onwards.

Now, the question is that, how manually we have seen that during the recovery of P 1 this is also forcing P J to recover all though P J was not faulty. Similarly PK also was triggered to be rolled back, so it is a cascaded rollback because, rollback of one process will create the orphan messages which in turn, will trigger the rollback of other non faulty processes. So, we are just seen through this particular example that, failure recovery is not a trivial task, it requires dependencies are to be captured and also it requires how the global state is to be recorded. So, that with the minimal loss the system can be restored after the failure

(Refer Slide Time: 20:58)



## Issues in failure recovery

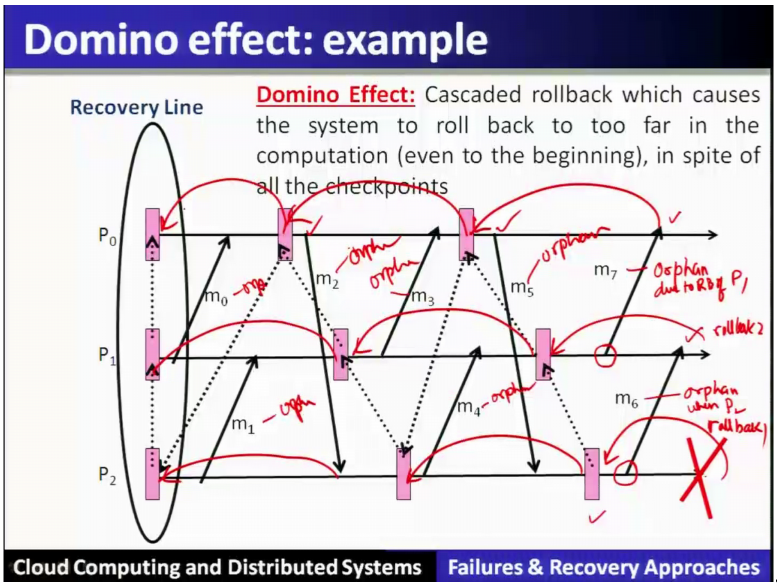
- The rollback of process $P_i$ to checkpoint $C_{i,1}$ created an orphan message H
- Orphan message I is created due to the roll back of process $P_j$ to checkpoint $C_{j,1}$
- Messages C, D, E, and F are potentially problematic

  -Message C: a delayed message
  -Message D: a lost message since the send event for D is recorded in the restored state for $P_j$, but the receive event has been undone at process $P_i$
  -Lost messages can be handled by having processes keep a message log of all the sent messages
  -Messages E, F: delayed orphan messages. After resuming execution from their checkpoints, processes will generate both of these messages

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Here issues in failure recovery. The rollback of a process P i to a checkpoint C i comma 1, created an orphan message H. Orphan message is created due to the rollback of a process P J to a previous to a checkpoint that is C J 1. So, the messages here we are seeing in the example C, D, E, F they are basically the problematic in that sense, that they have become the orphan message and it requires a rollback to undo that effect.

(Refer Slide Time: 21:44)


## Domino effect: example

**Domino Effect:** Cascaded rollback which causes the system to roll back to too far in the computation (even to the beginning), in spite of all the checkpoints

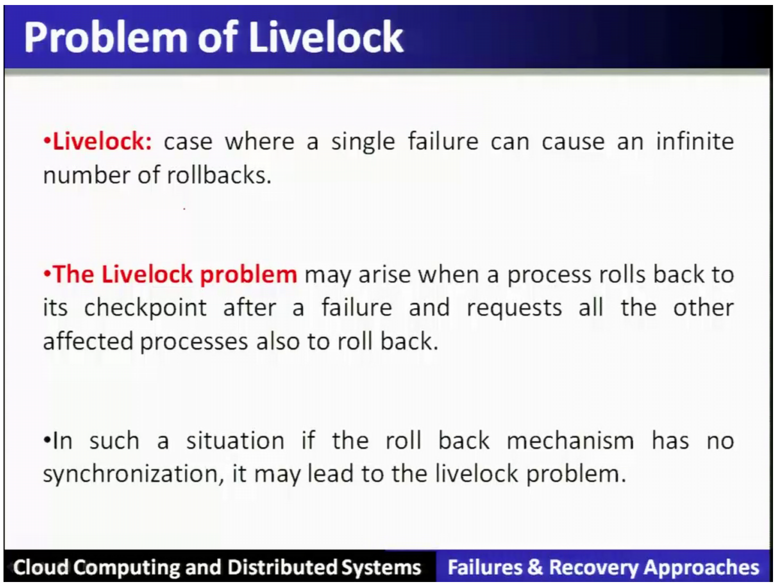
Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Closely the domino effect, that is called cascaded rollback which causes the system to rollback. So, let us that process P 2 is failed at this particular position. So, after the failure, it will recover from the previous checkpoint and this particular checkpoint will trigger the non faulty process P 1 also to rollback because, it has received this message m 6 whose send will be undone. So, that means, m 6 will become orphan message, when it will be when process P 2 will be rolled back

So, this particular orphan message will trigger to roll back to the previous consistent state. Now if it is rollback, then what will happen is m 7 which is received at P 0, but its send will be not there. So, this also will become orphan message due to the rollback of P 1. So, therefore, this will trigger P 1 to rollback from to the previous state. Now if P 0 is rollback, then what will happen to this particular message which is m 5, so m 5 will become orphan in this condition and we will trigger P 2 to further rollback to its previous state. Having roll back to the previous state this m 4 will become orphan therefore, it will also trigger P 1 2 further rollback.

Now, if P 1 is rollback then m 3 will become orphan and will trigger the P 0 also to further rollback. Having seen that P 0 is rollback then, this message M 2 will become orphan and this will trigger the further rollback. If this is rollback then, these will also rollback why because, m 1 will become orphan and if therefore, when this is rollback then m 0 will become orphan and finally, it will roll back to their previous state. So, this is an extreme condition of the domino effect.

(Refer Slide Time: 24:25)



## Problem of Livelock

•**Livelock:** case where a single failure can cause an infinite number of rollbacks.

•**The Livelock problem** may arise when a process rolls back to its checkpoint after a failure and requests all the other affected processes also to roll back.

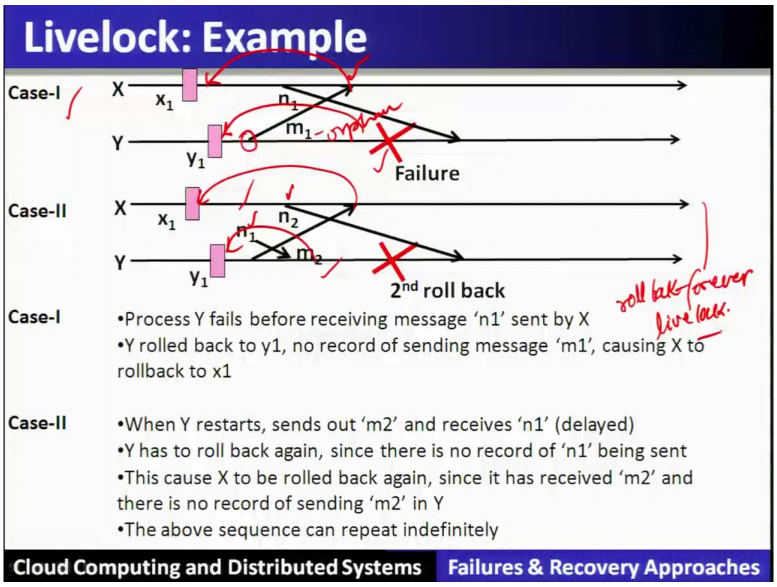
•In such a situation if the roll back mechanism has no synchronization, it may lead to the livelock problem.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Now, we will see another problem. We have seen earlier the problem of domino effect due to the orphan messages. Now we will see another problem that is called livelock problem of livelock.

The livelock case, where the single failure can cause an infinite number of rollback is called a livelock. The livelock problem may arise when a process roll back to its checkpoint after a failure and request all other affected processes also to roll back. In such a situation if rollback mechanism has no synchronization, which will lead to a infinite number of rollback this will lead to the livelock problem.

(Refer Slide Time: 25:16)


## Livelock: Example

Case-I
X
Y
$x_1$
$y_1$
$n_1$
$m_1$ - orphan
Failure
Case-II
X
Y
$x_1$
$y_1$
$n_1$
$n_2$
$m_2$
2nd roll back
roll back forever
live lock.

Case-I

- Process Y fails before receiving message 'n1' sent by X
- Y rolled back to y1, no record of sending message 'm1', causing X to rollback to x1

Case-II

- When Y restarts, sends out 'm2' and receives 'n1' (delayed)
- Y has to roll back again, since there is no record of 'n1' being sent
- This cause X to be rolled back again, since it has received 'm2' and there is no record of sending 'm2' in Y
- The above sequence can repeat indefinitely

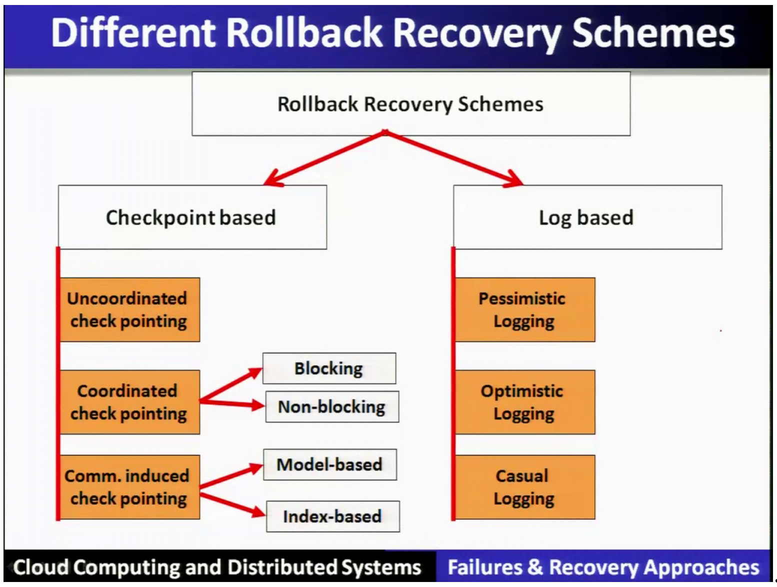
Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Let us see through this example. Here the case I says that, process Y fails because, the receiving message n 1 is sent by X. Y will be roll back to Y 1, no record of sending m 1 causing X to roll back to X 1. Let us understand this again.

In case I, you are seeing that the process Y is failed at this point of time, when it will recover then it will try to restart from this checkpoint. If it is restarting from this checkpoint or it is failing then, the send of this particular message m 1 will be lost, why because it is restarting at Y 1. Hence m 1 will become orphan message. So, if m 1 becomes orphan message and it is being received at a process X, this will trigger X to roll back at its checkpoint. Now we see that process X after it restarts, it sends another message n 2 meanwhile; the previous message n 1 also is received, that is called m 2.

Now, this particular message m 2, which is received has no sender, that is the sender of n 1 that is the send of a n 1 is not there. So, this particular message is orphan which will trigger the second rollback. And once it is rolled back then the received of this particular message, which is now becoming orphan this also will rollback. So, this particular process will continue to rollback forever, both of them will rollback forever and this is called a livelock situation.

(Refer Slide Time: 27:22)

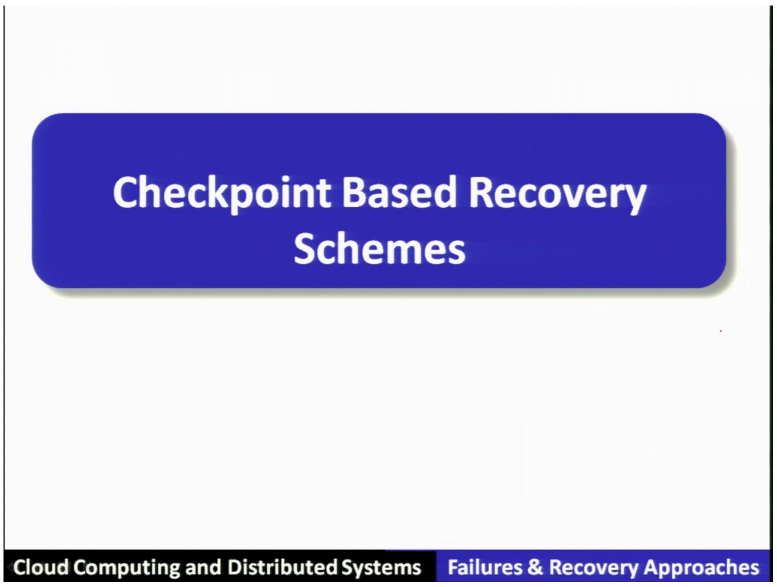


## Different Rollback Recovery Schemes

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Different rollback recovery schemes, the other one is called log based rollback recovery scheme. Log based rollback recovery is schemes are of three types: Pessimistic logging, optimistic logging and casual logging. We will see them in more details. Similarly checkpoint based rollback recovery scheme are also three types: Uncoordinated checkpointing then, coordinated checkpointing; coordinated checkpointing is also of two types blocking and non blocking and communication induced checkpointing is also there are two types; model based and index based checkpointing method.

(Refer Slide Time: 27:58)

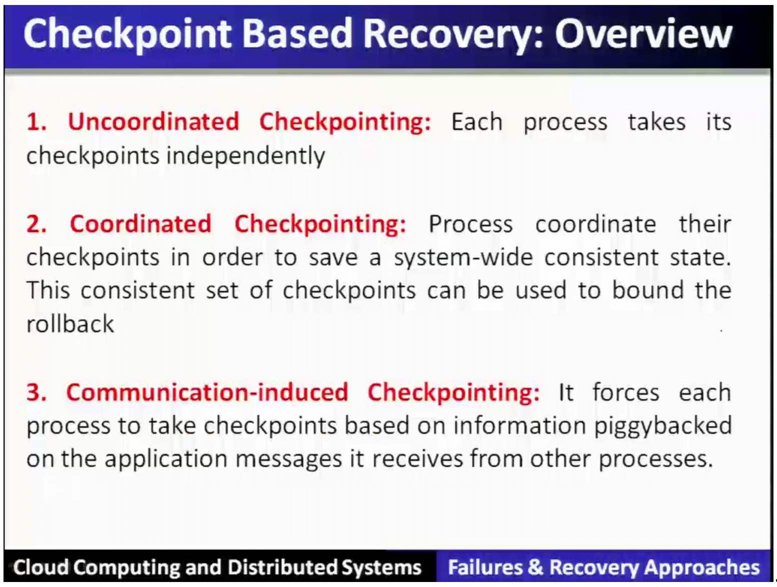


Checkpoint Based Recovery Schemes

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, checkpointing based recovery schemes.

(Refer Slide Time: 28:01)



## Checkpoint Based Recovery: Overview

1. **Uncoordinated Checkpointing:** Each process takes its checkpoints independently

2. **Coordinated Checkpointing:** Process coordinate their checkpoints in order to save a system-wide consistent state. This consistent set of checkpoints can be used to bound the rollback

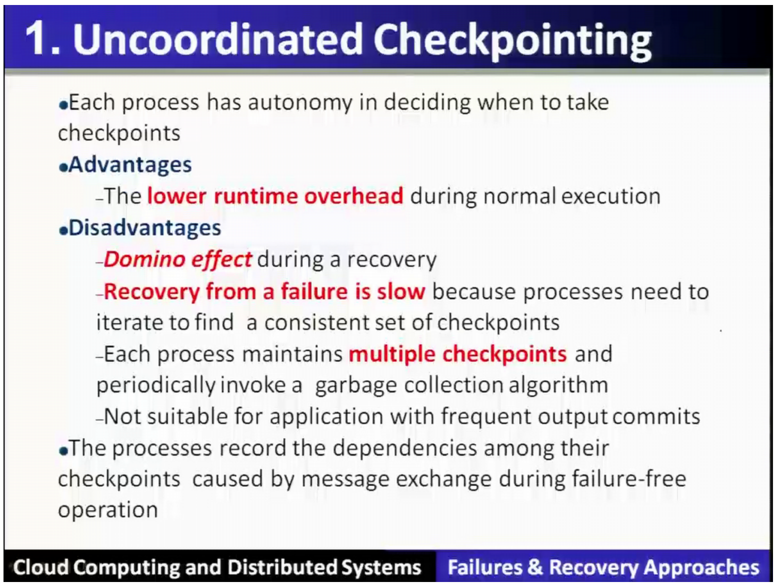
3. **Communication-induced Checkpointing:** It forces each process to take checkpoints based on information piggybacked on the application messages it receives from other processes.

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

So, as we have seen that they have the three different types, uncoordinated, coordinated and communication induced checkpointing. Uncoordinated checkpointing here the each process takes its checkpoints independently initial. This is quite simple, but the implications are more severe at the time of recovery. Coordinated checkpointing that is a processes, together they coordinate to take the checkpoints in order to capture the consistent global checkpoints. This, consistent global checkpoints can be useful for the rollback recovery without much effort

Third type of checkpointing is called communication induced pointing. It forces each process to take checkpoint based on the information which is piggybacked on the message application it will receive from the other processes. This kind of checkpoint we have already seen in a global state recording algorithm given by the lamport.

(Refer Slide Time: 29:08)



## 1. Uncoordinated Checkpointing

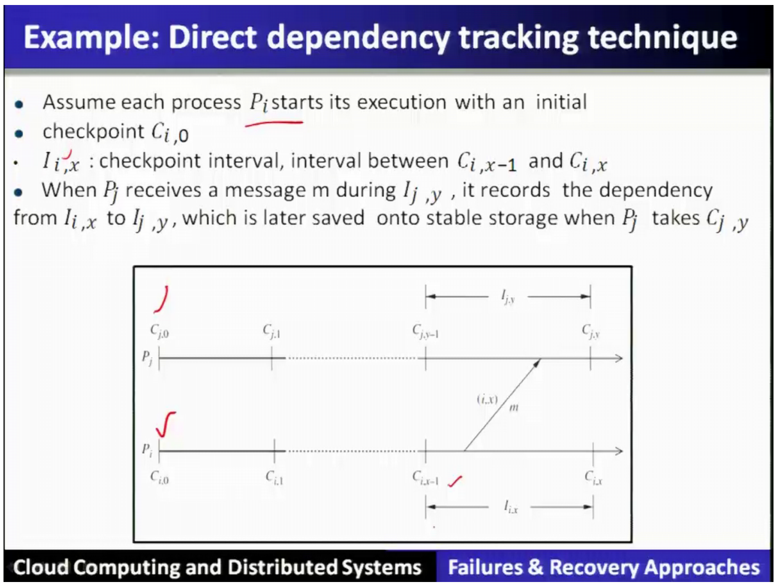
- Each process has autonomy in deciding when to take checkpoints
- **Advantages**
  - The **lower runtime overhead** during normal execution
- **Disadvantages**
  - ***Domino effect*** during a recovery
  - **Recovery from a failure is slow** because processes need to iterate to find a consistent set of checkpoints
  - Each process maintains **multiple checkpoints** and periodically invoke a garbage collection algorithm
  - Not suitable for application with frequent output commits
- The processes record the dependencies among their checkpoints caused by message exchange during failure-free operation

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

So, let us see the uncoordinated checkpointing. Here each process has autonomy in deciding when to take checkpoints. Advantage is that, it has lower runtime overhead during the normal course of execution, but the disadvantage is that this kind of uncoordinated checkpointing will suffer from domino effect during the recovery time.

So, recovery from failure is also slow why because, at the time of recovery they have to check and find out they have to search for a consistent set of checkpoints, because all local check points are uncoordinated, so now, at the time of recovery they have to do this. We will maintain a multiple checkpoints and periodically invoke a garbage collection algorithm. This kind of method is not suitable for application with frequent output commits.

(Refer Slide Time: 29:59)



## Example: Direct dependency tracking technique

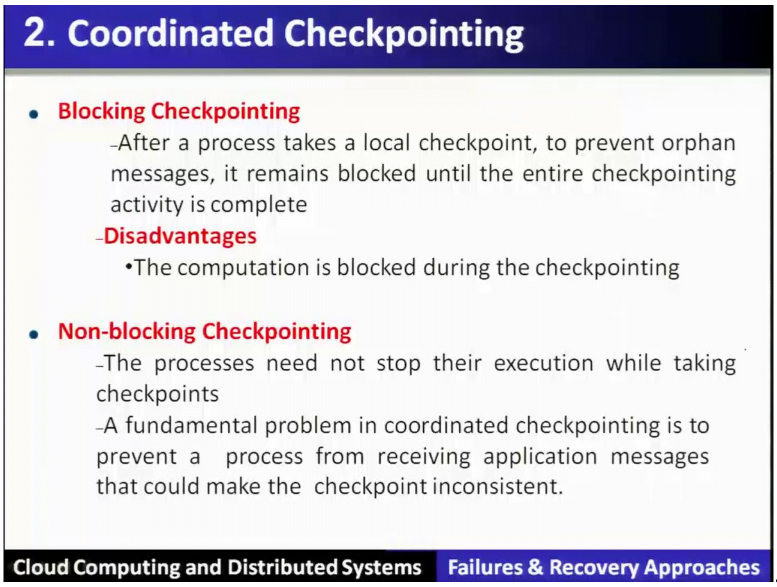
- Assume each process $P_i$ starts its execution with an initial
- checkpoint $C_{i,0}$
- $I_{i,x}$ : checkpoint interval, interval between $C_{i,x-1}$ and $C_{i,x}$
- When $P_j$ receives a message m during $I_{j,y}$ , it records the dependency from $I_{i,x}$ to $I_{j,y}$, which is later saved onto stable storage when $P_j$ takes $C_{j,y}$

Cloud Computing and Distributed Systems Failures & Recovery Approaches

The dependency, direct dependency tracking technique, which is shown here in this particular example, says that, let P i be the process, which is starts its execution by recording its initial checkpoint. Now i x is an interval between C i x comma x minus 1 and C i comma x the checkpoints between these two checkpoints this interval is called i x.

So, when P j process receives a message m during that the interval that is, j in j process the interval is y. It records the dependency from the interval of a process i xth interval to the interval of j yth interval and which is later stored into the stable storage when P j takes its checkpoint C j y.

(Refer Slide Time: 31:10)



## 2. Coordinated Checkpointing

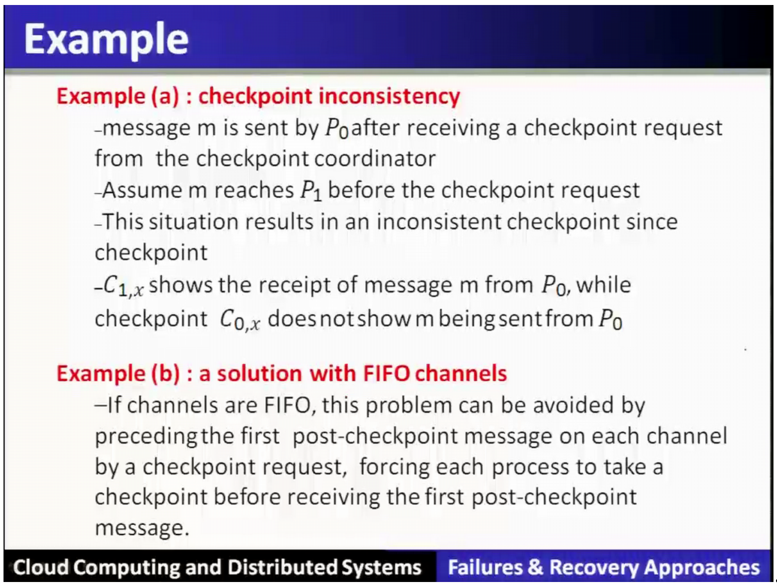
- **Blocking Checkpointing**
  - After a process takes a local checkpoint, to prevent orphan messages, it remains blocked until the entire checkpointing activity is complete
  - **Disadvantages**
    - The computation is blocked during the checkpointing
- **Non-blocking Checkpointing**
  - The processes need not stop their execution while taking checkpoints
  - A fundamental problem in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Blocking checkpointing: So, after a process takes a local checkpoint to prevent the orphan messages, it remains blocked until the entire checkpointing activity is complete. The disadvantage of blocking checkpointing is that the computation is blocked during the checkpointing process or a checkpointing duration.

Nonblocking checkpointing: The process need not stop their execution, while taking the checkpoints. The fundamental problem in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.

(Refer Slide Time: 31:50)



## Example

**Example (a) : checkpoint inconsistency**

- message m is sent by $P_0$ after receiving a checkpoint request from the checkpoint coordinator
- Assume m reaches $P_1$ before the checkpoint request
- This situation results in an inconsistent checkpoint since checkpoint
- $C_{1,x}$ shows the receipt of message m from $P_0$, while checkpoint $C_{0,x}$ does not show m being sent from $P_0$

**Example (b) : a solution with FIFO channels**

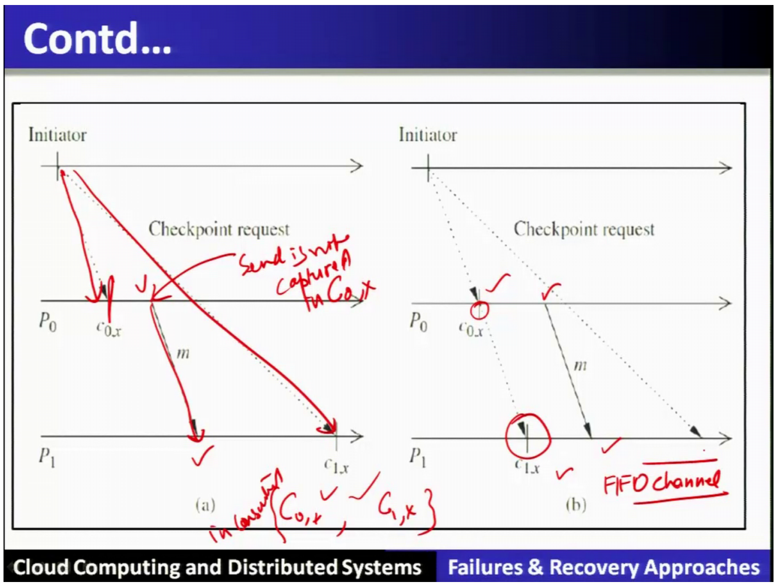
- If channels are FIFO, this problem can be avoided by preceding the first post-checkpoint message on each channel by a checkpoint request, forcing each process to take a checkpoint before receiving the first post-checkpoint message.

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

So, example, here the checkpoint inconsistency, that we have already seen that the message m is send by P 0, after receiving the checkpoint request from the checkpoint coordinator.

(Refer Slide Time: 32:04)

## Contd…



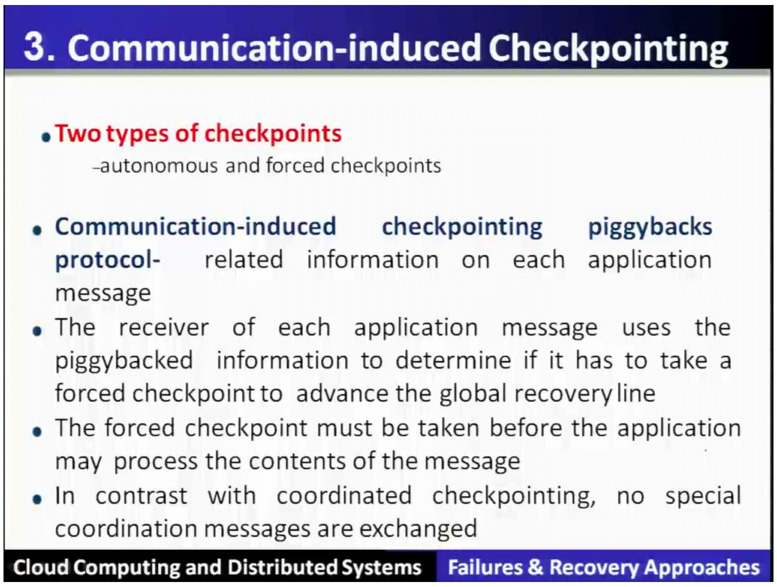
Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Here in this example. So, the checkpoint coordinator will send the message to take a checkpoint that will be received. So, assume m reached P 1 before the checkpoint request.

So, this particular m reaches here to P 1, before this particular before this request. This situation results in a inconsistent checkpoint, since the since the checkpoints c 1 comma x shows the receipt of a message m from p 0, while the checkpoint c 0 comma x does not show that m being sent from. So, here we can see this particular situation, in this checkpoint the send of m is not recorded, not captured in the checkpoint c 0 comma x; whereas, this receive of m is captured here in c 1 comma x. Hence this collection 0 comma x and this is an inconsistent checkpoint.

Now, another example shows that if, a FIFO channel is considered, if the channels are FIFO this problem can be avoided by preceding the first post checkpoint message on each channel by the checkpoint request, forcing each process to take a checkpoint before receiving the first post checkpoint message. So, here in this particular situation, so, after taking a checkpoint, it assumes a FIFO (Refer Time: 33:56) This particular checkpoint message will reach at this instant. So, therefore, the message which is sent after taking checkpoint at P 0, will also be received at a later point of time. So, this is possible if we assume a FIFO channel, in this particular situation.

(Refer Slide Time: 34:27)



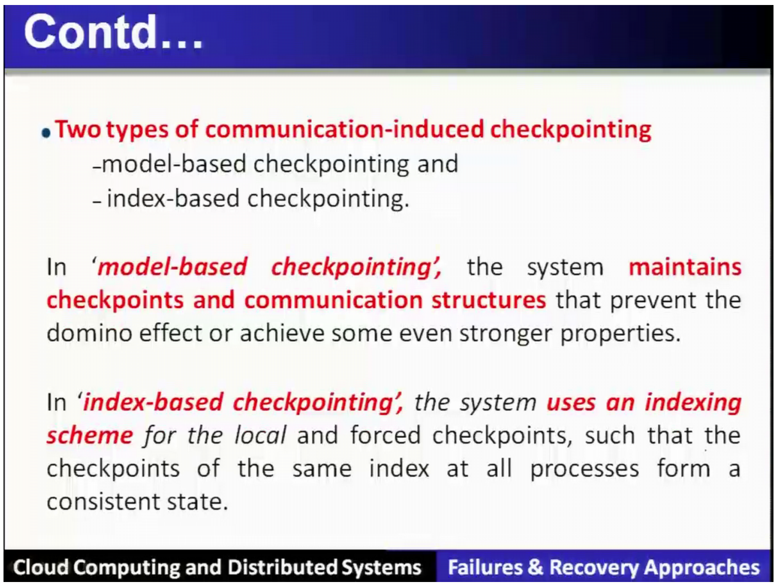
## 3. Communication-induced Checkpointing

- Two types of checkpoints
  - autonomous and forced checkpoints
- Communication-induced checkpointing piggybacks protocol- related information on each application message
- The receiver of each application message uses the piggybacked information to determine if it has to take a forced checkpoint to advance the global recovery line
- The forced checkpoint must be taken before the application may process the contents of the message
- In contrast with coordinated checkpointing, no special coordination messages are exchanged

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

The third type of checkpointing is called communication induced checkpointing. This is also of two types of, autonomous and forced checkpoint. So, communication induced checkpointing, the piggybacks protocol, that is the related information on the application messages will be send the receiver of each message uses the piggyback information to determine, if it has to take a forced checkpoint or to advance the global recovery line. The fourth checkpoint must be taken before the application may process the content of the message. In contrast with the coordinated checkpointing, no special coordination messages are exchanged here in this particular situation.

(Refer Slide Time: 35:10)



## Contd…

- **Two types of communication-induced checkpointing**
  - model-based checkpointing and
  - index-based checkpointing.

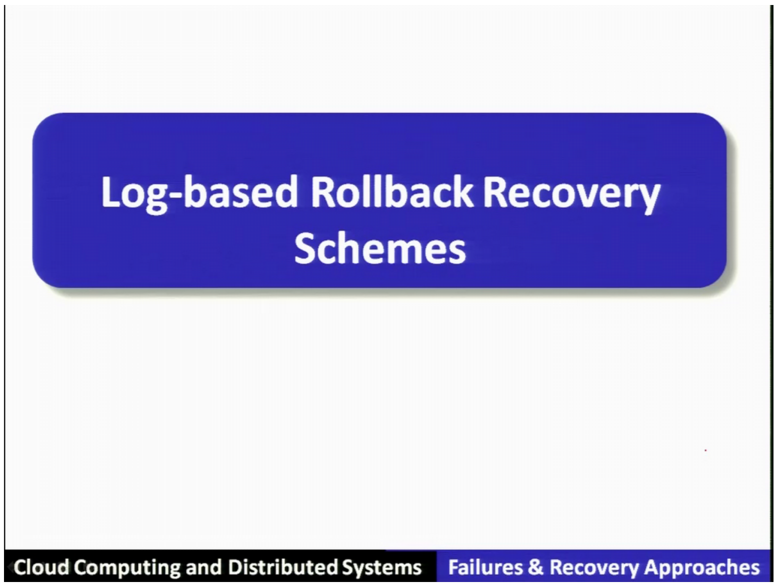
In ***'model-based checkpointing'***, the system **maintains checkpoints and communication structures** that prevent the domino effect or achieve some even stronger properties.

In ***'index-based checkpointing'***, *the system* ***uses an indexing scheme*** *for the local* and forced checkpoints, such that the checkpoints of the same index at all processes form a consistent state.

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

There are two types of communication: Induced checkpointing, model based checkpointing and indexed based checkpointing. In model based checkpointing, the system maintains the checkpoints and communication structuresm that prevents the domino effect or to achieve some even stronger properties. In index based checkpointing, the system uses an indexing scheme for local and forced checkpoints such that, the checkpoints of the same index at all the processes forms a consistent state.

(Refer Slide Time: 35:39)

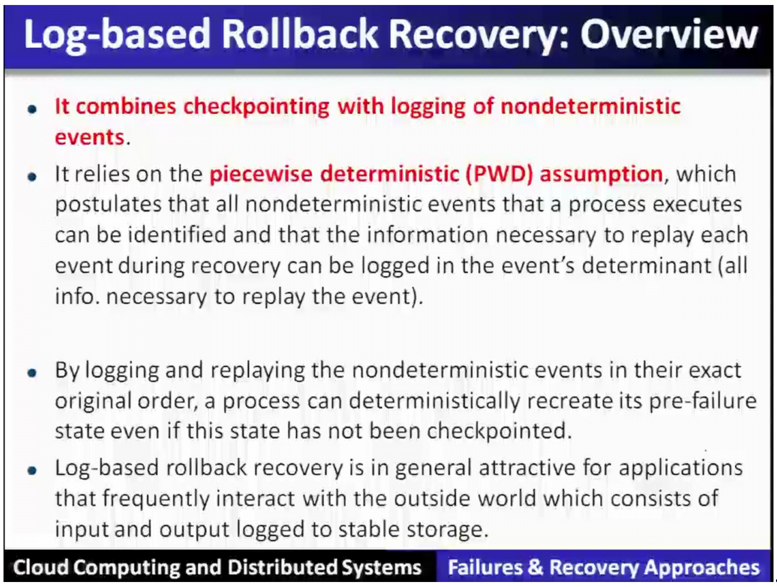


## Log-based Rollback Recovery Schemes

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Let us see the log based rollback recovery scheme.

(Refer Slide Time: 35:43)



## Log-based Rollback Recovery: Overview

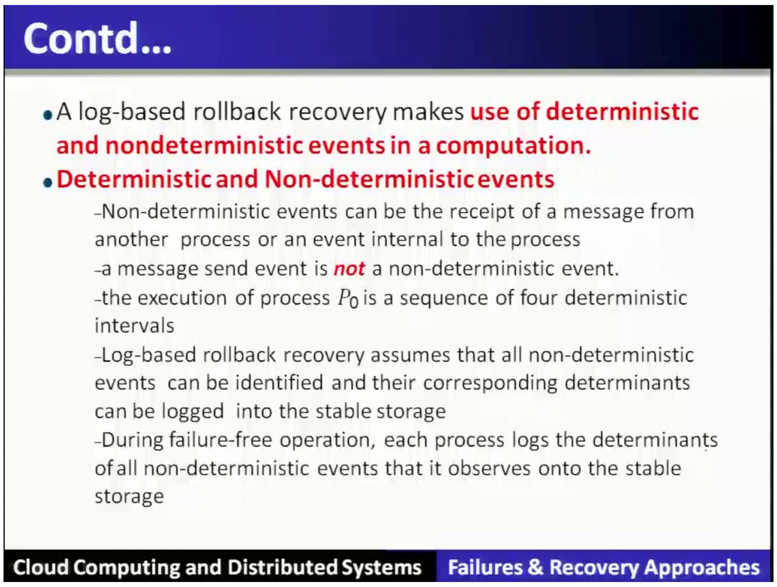
- It combines checkpointing with logging of nondeterministic events.
- It relies on the piecewise deterministic (PWD) assumption, which postulates that all nondeterministic events that a process executes can be identified and that the information necessary to replay each event during recovery can be logged in the event's determinant (all info. necessary to replay the event).
- By logging and replaying the nondeterministic events in their exact original order, a process can deterministically recreate its pre-failure state even if this state has not been checkpointed.
- Log-based rollback recovery is in general attractive for applications that frequently interact with the outside world which consists of input and output logged to stable storage.

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

So, it combines the checkpointing with the logging of nondeterministic events. It relies on piecewise deterministic assumptions, which postulate that all nondeterministic events, that a process executes can be identified and that the information necessary to replay each event during the recovery can be logged in the events determinants. By logging and replaying the nondeterministic event in their exact original order, the process can deterministically recreate its pre failure state even if, by state has not been checkpointed.

So, log based rollback recovery is in general attractive for the application, that frequently interact with the outside world, which consists of input and output logged to the stable storage.

(Refer Slide Time: 36:35)



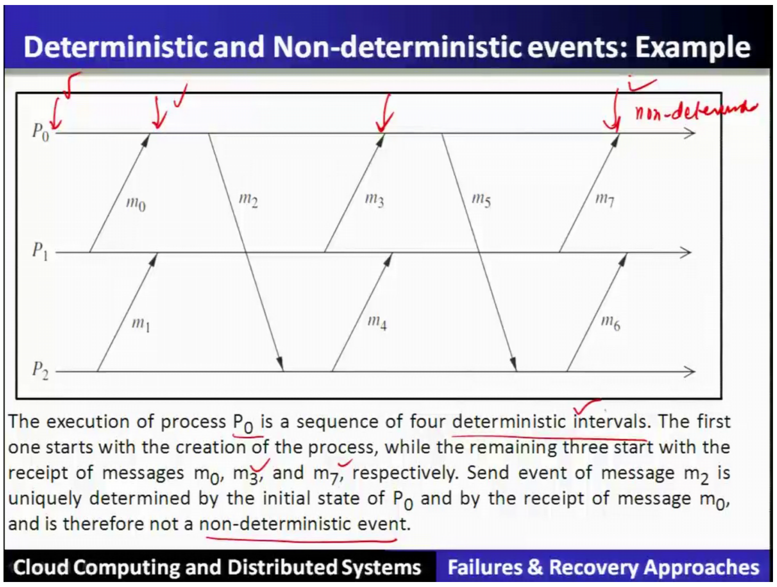
## Contd...

- A log-based rollback recovery makes **use of deterministic and nondeterministic events in a computation.**
- **Deterministic and Non-deterministic events**
  - Non-deterministic events can be the receipt of a message from another process or an event internal to the process
  - a message send event is ***not*** a non-deterministic event.
  - the execution of process $P_0$ is a sequence of four deterministic intervals
  - Log-based rollback recovery assumes that all non-deterministic events can be identified and their corresponding determinants can be logged into the stable storage
  - During failure-free operation, each process logs the determinants of all non-deterministic events that it observes onto the stable storage

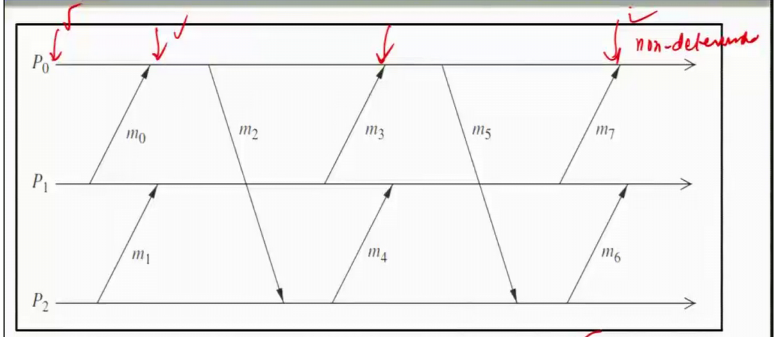
Cloud Computing and Distributed Systems | Failures & Recovery Approaches

So, log based rollback recovery makes use of, deterministic and nondeterministic events in the computation. So, deterministic and nondeterministic events, so nondeterministic event can be the receipt of a message from another process or the event, which is internal to a to a process. A message send is not a nondeterministic event. The execution of a process P 0 is a sequence of four deterministic events in the following figure

(Refer Slide Time: 37:05)



## Deterministic and Non-deterministic events: Example



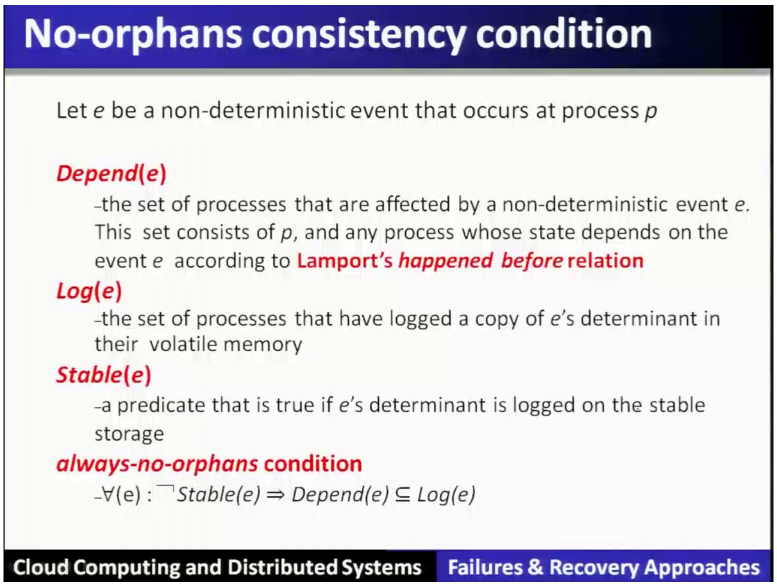
The execution of process $P_0$ is a sequence of four deterministic intervals. The first one starts with the creation of the process, while the remaining three start with the receipt of messages $m_0$, $m_3$, and $m_7$, respectively. Send event of message $m_2$ is uniquely determined by the initial state of $P_0$ and by the receipt of message $m_0$, and is therefore not a non-deterministic event.

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

So, here this is the creation of this event is a nondeterministic event. Then receive of the message m 0 is also a nondeterministic event. The receipt of a message m 3 is also nondeterministic event and the receipt of a message m 7 is also nondeterministic event So, here it is shown that the execution of process P 0 is a sequence of four deterministic event. The first one starts the creation of a process. These are the four deterministic, so these are the nondeterministic events and in this example it is shown as four different intervals of a deterministic event.

The first one starts at the creation of a process that I have already told, while the remaining three start with the receipt of a message m 0 m 3 and m 7 respectively. Send event of a message m 2, this is not is uniquely determined by the initial state of P 0 and by the receipt of a message m 0. Therefore, it is not a nondeterministic event. So, these are all only four are nondeterministic events, which will divide into four deterministic intervals.

(Refer Slide Time: 38:28)



## No-orphans consistency condition

Let *e* be a non-deterministic event that occurs at process *p*

***Depend*(*e*)**

-the set of processes that are affected by a non-deterministic event *e*. This set consists of *p*, and any process whose state depends on the event *e* according to **Lamport's *happened before* relation**

***Log*(*e*)**

-the set of processes that have logged a copy of *e*'s determinant in their volatile memory

***Stable*(*e*)**

-a predicate that is true if *e*'s determinant is logged on the stable storage

***always-no-orphans* condition**

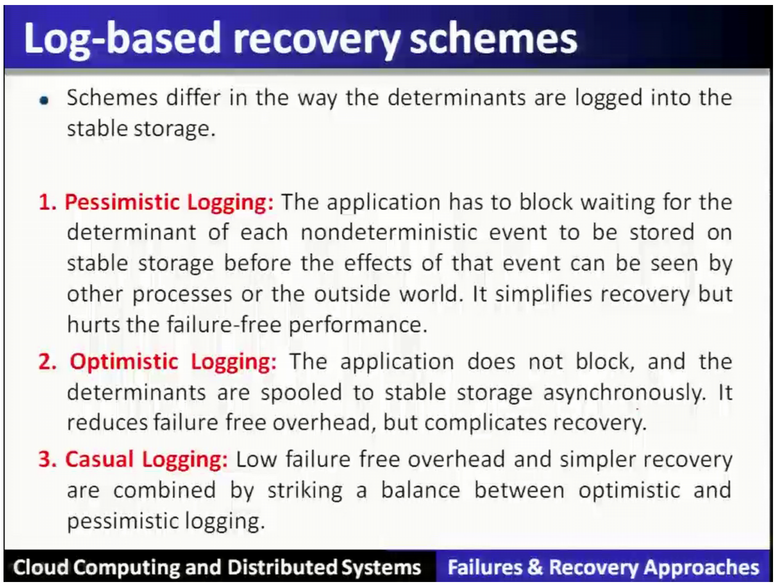
-$\forall(e) : \neg Stable(e) \Rightarrow Depend(e) \subseteq Log(e)$

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

No orphan consistency conditions, let e be a nondeterministic event, that occurs at a process, p depend e a set of processes that are affected by a nondeterministic event, is denoted by this. This set consist of p i and any process whose state depends on the event a according to the lamports happened before relation; log e, the set of processes that have logged a copy of e’s determinants, in their volatile memory.

Stable e a predicate that is true, if e is determinant is logged on the stable storage. Always no orphan conditions says that for all e, not orph stable e implies that it is depend e is a subset of log e; that means, it is there in the log e.

(Refer Slide Time: 39:24)



## Log-based recovery schemes

- Schemes differ in the way the determinants are logged into the stable storage.

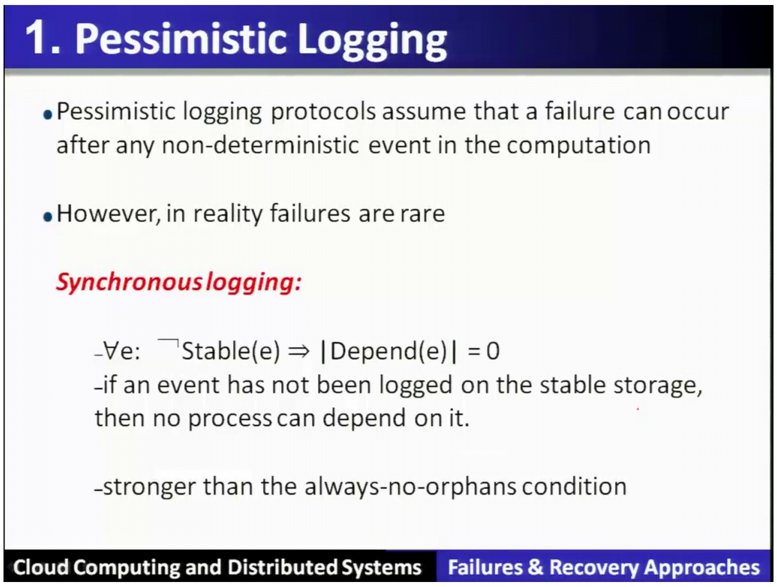
1. **Pessimistic Logging:** The application has to block waiting for the determinant of each nondeterministic event to be stored on stable storage before the effects of that event can be seen by other processes or the outside world. It simplifies recovery but hurts the failure-free performance.
2. **Optimistic Logging:** The application does not block, and the determinants are spooled to stable storage asynchronously. It reduces failure free overhead, but complicates recovery.
3. **Casual Logging:** Low failure free overhead and simpler recovery are combined by striking a balance between optimistic and pessimistic logging.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, log based recovery schemes differ in the way determinants are logged onto the stable storage. Therefore, it is called as pessimistic logging optimistic and casual logging. The application has to block waiting for the determinant of each nondeterministic event to be stored on the stable storage before the effects of that event can be seen by the other processes or the outside world. It simplifies the recovery, but hurts the failure free performance.

Optimistic logging the application does not block and determines and the determinants are spooled to the stable storage asynchronously. It reduces the failure free overhead, but complicates the recovery method. Casual logging is, low failure free overhead simpler recovery is combined by striking a balance between optimistic and pessimistic.

(Refer Slide Time: 40:18)



## 1. Pessimistic Logging

- Pessimistic logging protocols assume that a failure can occur after any non-deterministic event in the computation
- However, in reality failures are rare

***Synchronous logging:***

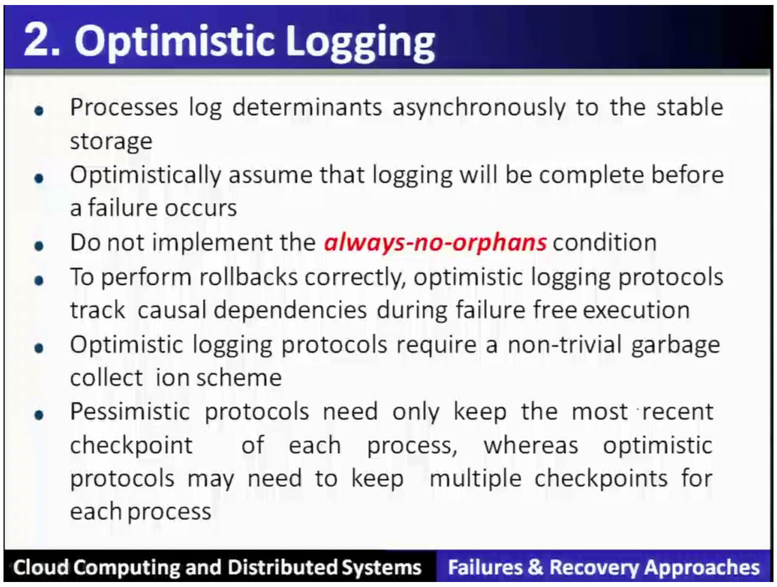
-$\forall e$: $\neg$Stable(e) $\Rightarrow$ |Depend(e)| = 0
-if an event has not been logged on the stable storage, then no process can depend on it.

-stronger than the always-no-orphans condition

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, let us see the pessimistic; pessimistic logging protocol, assume a failure can occur after any nondeterministic event in the computation. However, in reality these failures are rare. So, this is shown in the example

(Refer Slide Time: 40:35)



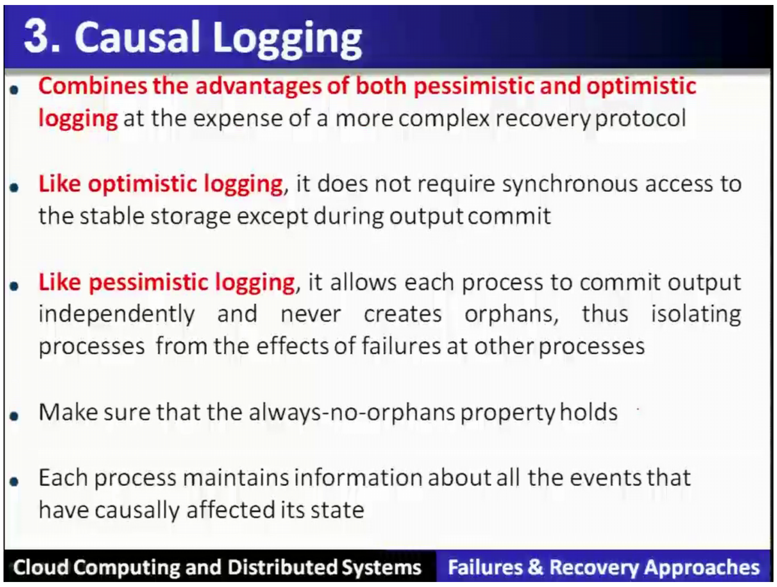
## 2. Optimistic Logging

- Processes log determinants asynchronously to the stable storage
- Optimistically assume that logging will be complete before a failure occurs
- Do not implement the ***always-no-orphans*** condition
- To perform rollbacks correctly, optimistic logging protocols track causal dependencies during failure free execution
- Optimistic logging protocols require a non-trivial garbage collect ion scheme
- Pessimistic protocols need only keep the most recent checkpoint of each process, whereas optimistic protocols may need to keep multiple checkpoints for each process

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Now, another scheme is called optimistic logging, here the processes log determinants asynchronously to the stable storage. So, optimistically assumes that, logging will be complete before the failure occurs. This is also shown in the example.

(Refer Slide Time: 40:52)



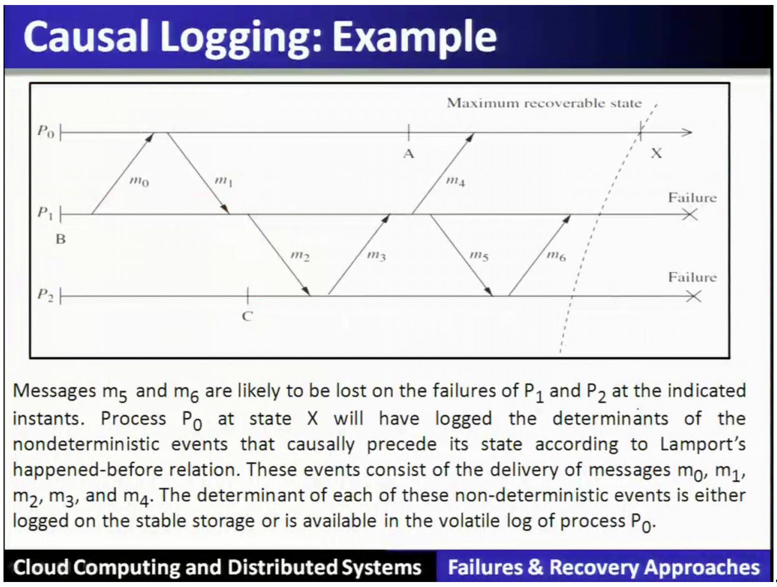
## 3. Causal Logging

- Combines the advantages of both pessimistic and optimistic logging at the expense of a more complex recovery protocol
- Like optimistic logging, it does not require synchronous access to the stable storage except during output commit
- Like pessimistic logging, it allows each process to commit output independently and never creates orphans, thus isolating processes from the effects of failures at other processes
- Make sure that the always-no-orphans property holds
- Each process maintains information about all the events that have causally affected its state

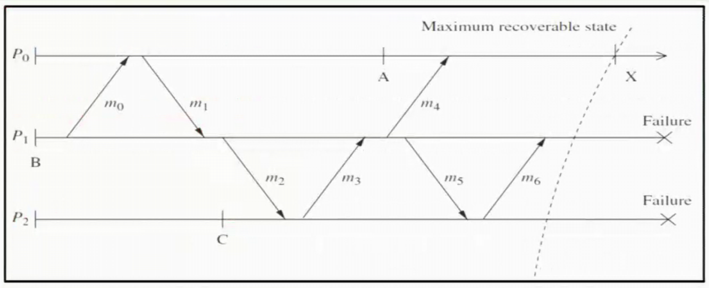
Cloud Computing and Distributed Systems Failures & Recovery Approaches

Third one is called casual logging; it combines the advantage of both pessimistic and optimistic locking, at the expense of more complicated recovery protocols.

(Refer Slide Time: 41:03)



## Causal Logging: Example

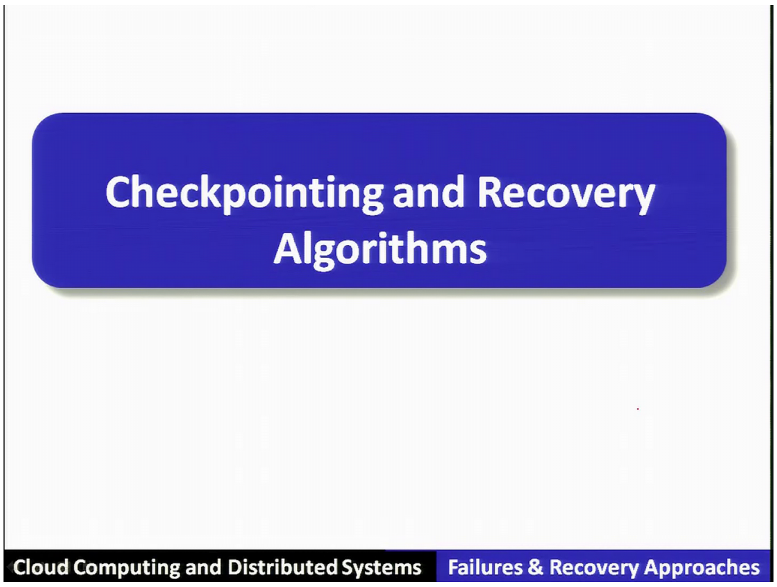


Messages $m_5$ and $m_6$ are likely to be lost on the failures of $P_1$ and $P_2$ at the indicated instants. Process $P_0$ at state X will have logged the determinants of the nondeterministic events that causally precede its state according to Lamport's happened-before relation. These events consist of the delivery of messages $m_0$, $m_1$, $m_2$, $m_3$, and $m_4$. The determinant of each of these non-deterministic events is either logged on the stable storage or is available in the volatile log of process $P_0$.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

This also is shown in this running example.

(Refer Slide Time: 41:06)

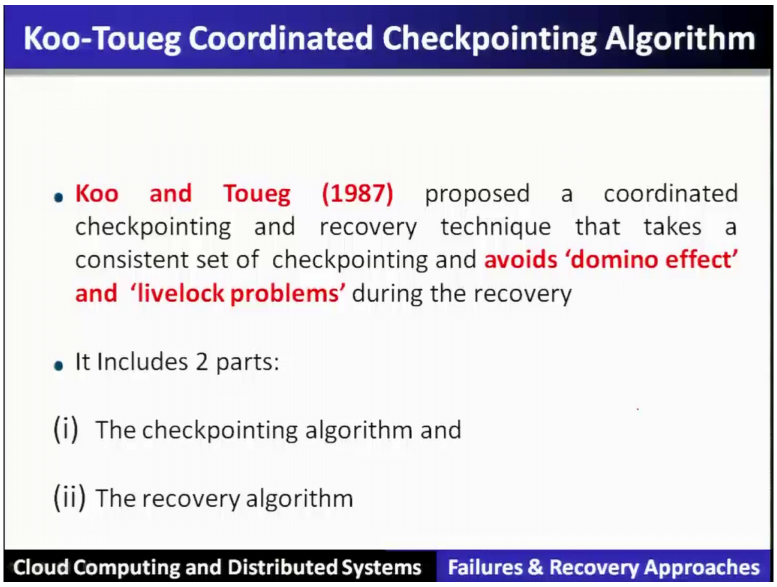


## Checkpointing and Recovery Algorithms

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Checkpointing and recovery algorithms.

(Refer Slide Time: 41:08)



## Koo-Toueg Coordinated Checkpointing Algorithm

- **Koo and Toueg (1987)** proposed a coordinated checkpointing and recovery technique that takes a consistent set of checkpointing and **avoids 'domino effect' and 'livelock problems'** during the recovery
- It Includes 2 parts:

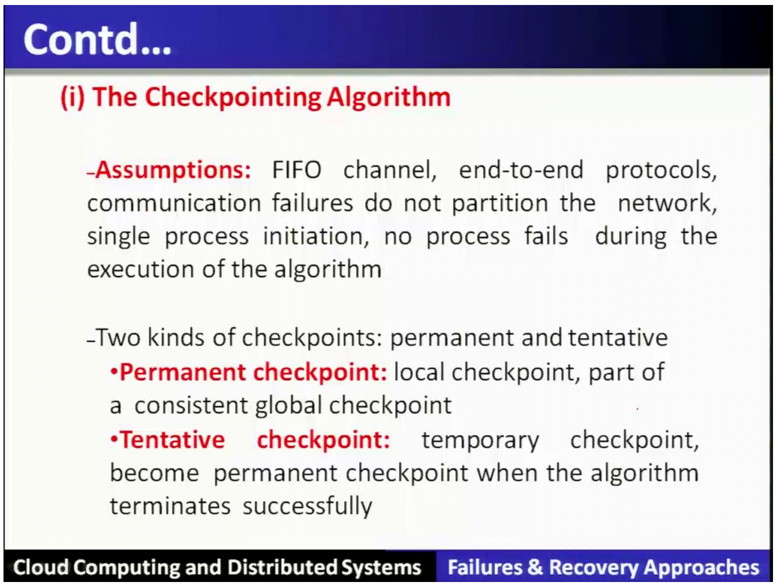
(i) The checkpointing algorithm and

(ii) The recovery algorithm

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Koo and Toueg coordinated checkpointing algorithm, this algorithm is given in 1987. This is a proposed for coordinated checkpointing and rollback recovery techniques, that takes constant set of checkpoints checkpointing and avoid the domino effect and the livelock problem, during the recovery in the algorithm. So, this algorithm has two parts; the first part is called checkpointing algorithm, the second part is called recovery algorithm.

(Refer Slide Time: 41:39)



## Contd...

**(i) The Checkpointing Algorithm**

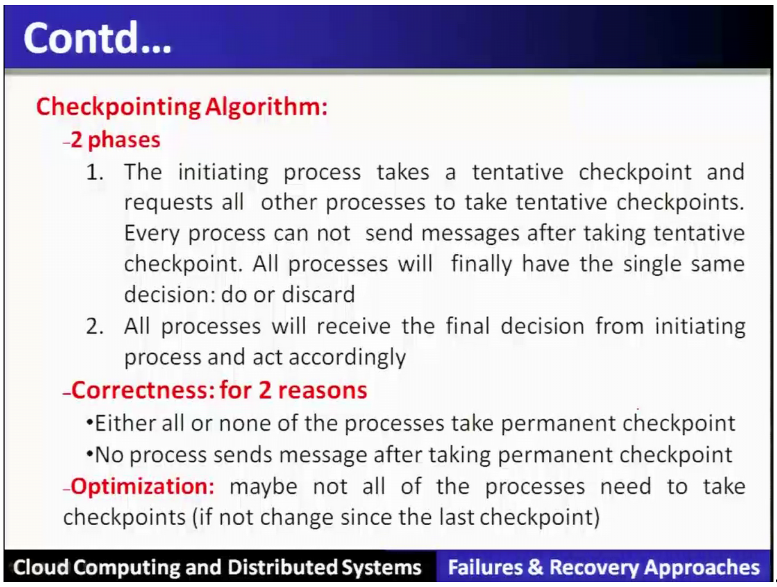
- **Assumptions:** FIFO channel, end-to-end protocols, communication failures do not partition the network, single process initiation, no process fails during the execution of the algorithm
- Two kinds of checkpoints: permanent and tentative
  - **Permanent checkpoint:** local checkpoint, part of a consistent global checkpoint
  - **Tentative checkpoint:** temporary checkpoint, become permanent checkpoint when the algorithm terminates successfully

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

The checkpointing algorithm assumes the FIFO channel and the end to end protocol communication failures do not partition the network. A single process will initiate this algorithm and no process fails during the execution of the algorithm.

Now, this algorithm checkpointing takes two kinds of checkpoints that is, the first one is called the tentative checkpoint and then it will be made permanent checkpoint. Tentative checkpoint is a temporary checkpoint, will become permanent checkpoint when the algorithm terminates successfully. A permanent checkpoint is a local checkpoint, part of the consistent global checkpoint.

(Refer Slide Time: 42:20)



## Contd...

**Checkpointing Algorithm:**

- **2 phases**
  1. The initiating process takes a tentative checkpoint and requests all other processes to take tentative checkpoints. Every process can not send messages after taking tentative checkpoint. All processes will finally have the single same decision: do or discard
  2. All processes will receive the final decision from initiating process and act accordingly
- **Correctness: for 2 reasons**
  - Either all or none of the processes take permanent checkpoint
  - No process sends message after taking permanent checkpoint
- **Optimization:** maybe not all of the processes need to take checkpoints (if not change since the last checkpoint)

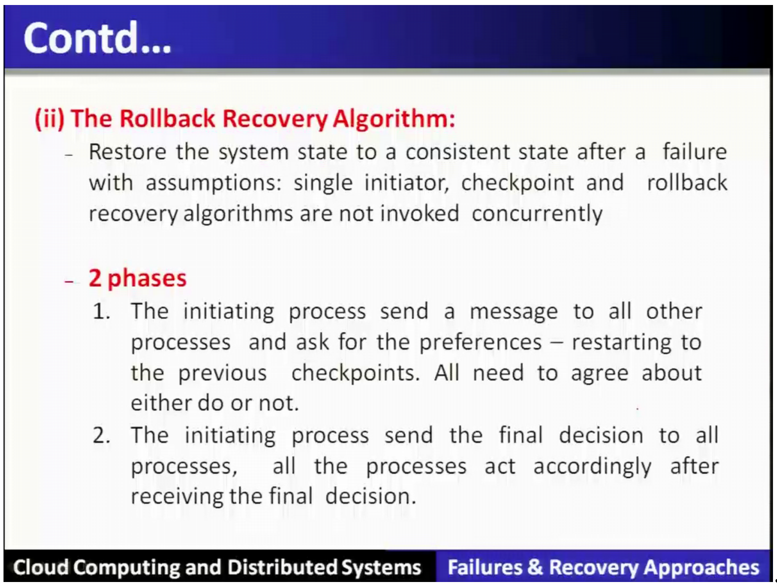
Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Let us see the checkpointing algorithm. It comprises of two phases; the first phase says that the initiating process takes a tentative checkpoint and requests all other process to take the tentative checkpoints.

Every process cannot send message, after taking the tentative checkpoint. All processes will finally, have a single same decision do or discard. The second phase says that, all processes will receive the final decision from initiating process and act accordingly. So, this algorithm is correct for a two reason, either all or none of the processes will take the permanent checkpoint and no process sends the message after taking a permanent checkpoint. Hence this algorithm is correct.

There are some optimizations, maybe not all process need to take the checkpoints. So, if there is no change in the state since, the last checkpoint, so therefore, there is a possibility that some of the process may not required to take a checkpoint. Hence that optimization if, it is included then it will also reduce the overhead of checkpointing algorithm.

(Refer Slide Time: 43:30)



## Contd…

**(ii) The Rollback Recovery Algorithm:**

- Restore the system state to a consistent state after a failure with assumptions: single initiator, checkpoint and rollback recovery algorithms are not invoked concurrently

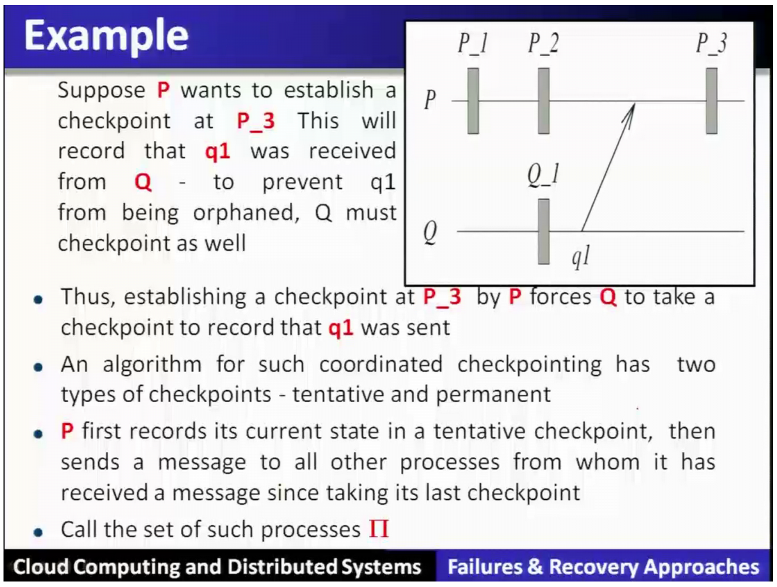
- **2 phases**
  1. The initiating process send a message to all other processes and ask for the preferences – restarting to the previous checkpoints. All need to agree about either do or not.
  2. The initiating process send the final decision to all processes, all the processes act accordingly after receiving the final decision.

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

The second part of this algorithm is called, rollback recovery algorithm. It restores the system state to a consistent state after failure with the assumption that; it is a single initiator checkpoint and rollback recovery algorithm are not in invoked concurrently.

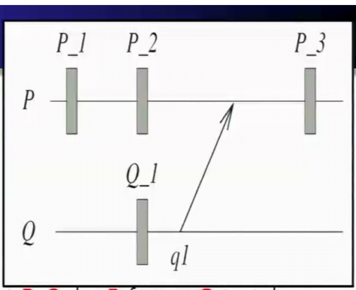
Rollback recovery algorithm also of two phases: The first phase says that, the initiating process will send a message to all other process and ask for the preferences, that is restarting to a previous checkpoints all need to agree about either do or not. So, the initiating process will send the final decision to all the processes.

(Refer Slide Time: 44:11)



## Example

Suppose P wants to establish a checkpoint at P_3 This will record that q1 was received from Q - to prevent q1 from being orphaned, Q must checkpoint as well

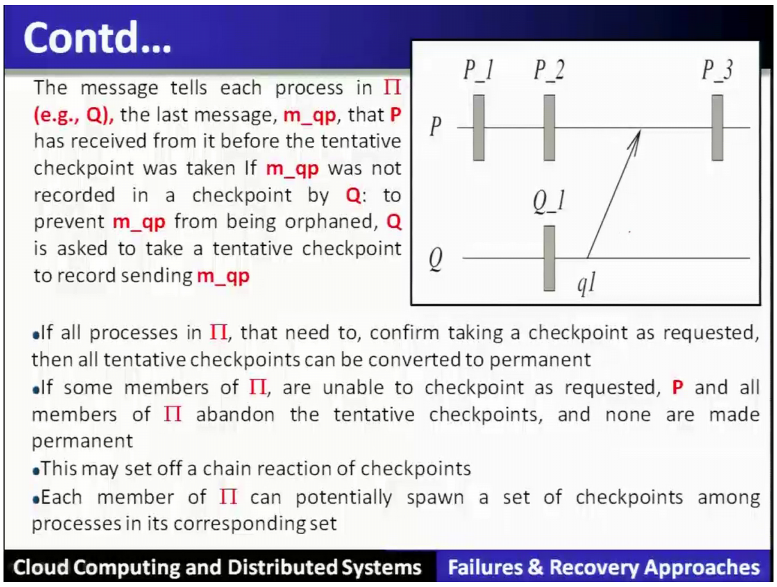


- Thus, establishing a checkpoint at P_3 by P forces Q to take a checkpoint to record that q1 was sent
- An algorithm for such coordinated checkpointing has two types of checkpoints - tentative and permanent
- P first records its current state in a tentative checkpoint, then sends a message to all other processes from whom it has received a message since taking its last checkpoint
- Call the set of such processes $\Pi$

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

All the processes act accordingly, after receiving the final decision. Here this is an example. Here suppose P wants to establish a checkpoint at P 3, this will record that q1 was received from q, to prevent from being orphaned q must checkpoint as well. Thus establishing the checkpoints at P 3 by P, forces q to take a checkpoint to record that q was sent. An algorithm for such coordinative checkpointing has two types of checkpoint: Tentative and permanent. So, here then, it will send a message to all the process from whom, it has received the message since, taking its last checkpoint, call that such a process.

(Refer Slide Time: 44:54)



## Contd...

The message tells each process in Π (e.g., Q), the last message, m_qp, that P has received from it before the tentative checkpoint was taken If m_qp was not recorded in a checkpoint by Q: to prevent m_qp from being orphaned, Q is asked to take a tentative checkpoint to record sending m_qp

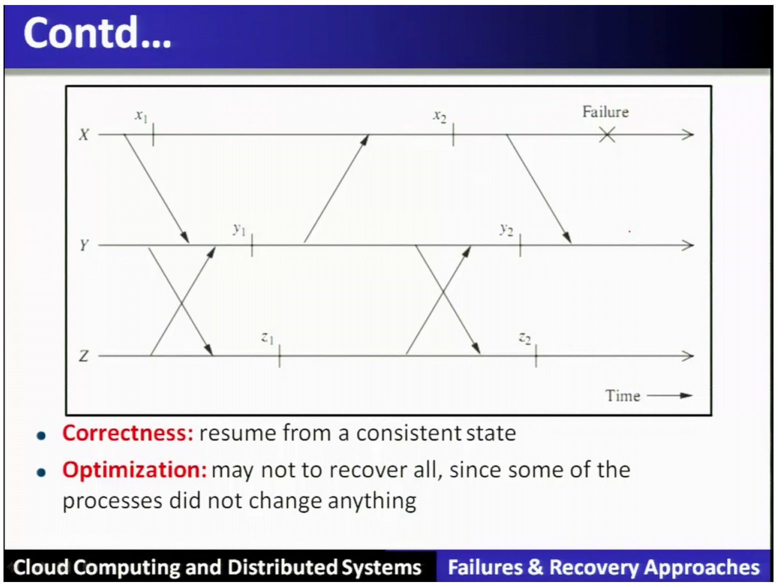
P_1
P_2
P_3
P
Q_1
Q
ql

- If all processes in Π, that need to, confirm taking a checkpoint as requested, then all tentative checkpoints can be converted to permanent
- If some members of Π, are unable to checkpoint as requested, P and all members of Π abandon the tentative checkpoints, and none are made permanent
- This may set off a chain reaction of checkpoints
- Each member of Π can potentially spawn a set of checkpoints among processes in its corresponding set

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, the message tells each process, the last message m pq that p has received from it tentative checkpoint and recorded the checkpoint by Q to prevent m p that message Q 1 from being orphaned and q is asked to take a tentative checkpoint, to record sending m 1. If all the processes in the checkpoint need to confirm as requested, then this particular tentative checkpoint can be converted to the permanent checkpoint.

(Refer Slide Time: 45:23)



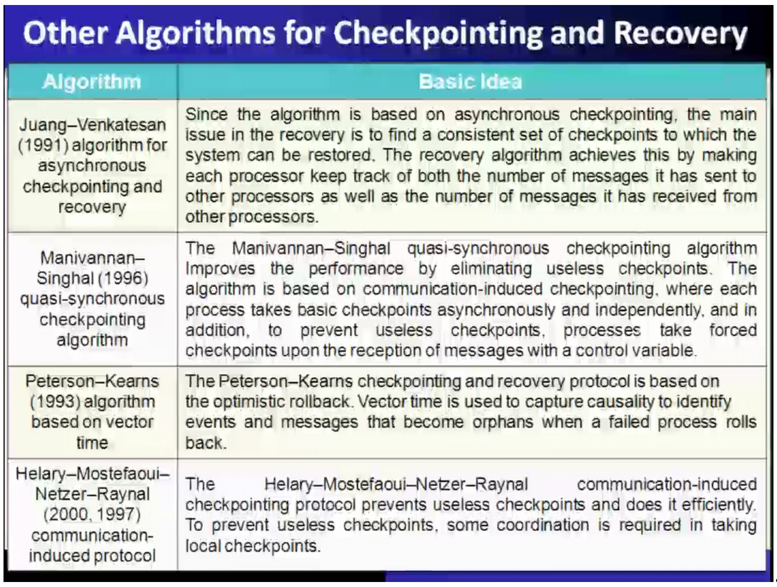
## Contd...

- **Correctness:** resume from a consistent state
- **Optimization:** may not to recover all, since some of the processes did not change anything

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Now, as far as the correctness is concerned, we can see here that will be resume from the consistent state. There is some optimizations why because, it is not that all processes are required to be recovered for example; since the last checkpoint there is no activity. Hence this particular process should not be rolled back. So, these are the few optimizations which are possible in this algorithm.

(Refer Slide Time: 45:48)



## Other Algorithms for Checkpointing and Recovery

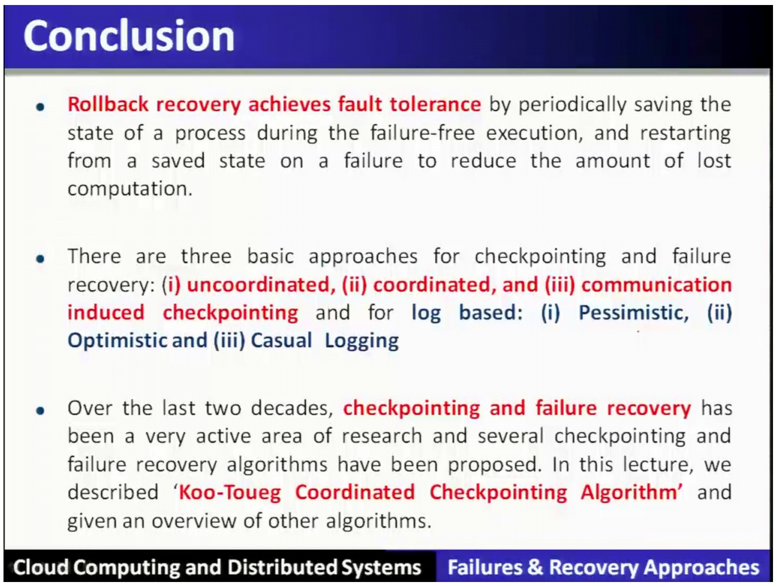
| Algorithm | Basic Idea |
|---|---|
| Juang–Venkatesan (1991) algorithm for asynchronous checkpointing and recovery | Since the algorithm is based on asynchronous checkpointing, the main issue in the recovery is to find a consistent set of checkpoints to which the system can be restored. The recovery 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. |
| Manivannan–Singhal (1996) quasi-synchronous checkpointing algorithm | The Manivannan–Singhal quasi-synchronous checkpointing algorithm Improves the performance by eliminating useless checkpoints. The algorithm is based on communication-induced checkpointing, where each process takes basic checkpoints asynchronously and independently, and in addition, to prevent useless checkpoints, processes take forced checkpoints upon the reception of messages with a control variable. |
| Peterson–Kearns (1993) algorithm based on vector time | The Peterson–Kearns checkpointing and recovery protocol is based on the optimistic rollback. Vector time is used to capture causality to identify events and messages that become orphans when a failed process rolls back. |
| Helary–Mostefaoui–Netzer–Raynal (2000, 1997) communication-induced protocol | The Helary–Mostefaoui–Netzer–Raynal communication-induced checkpointing protocol prevents useless checkpoints and does it efficiently. To prevent useless checkpoints, some coordination is required in taking local checkpoints. |

.

There are other algorithm for checkpointing and rollback recovery, which are shown here, Juang and Venkatesan, Manivannan and Single, Peterson and Kearns, Helary and

Mostefaoui. All these algorithms deals with the checkpointing and rollback recovery with some heuristics involved.

(Refer Slide Time: 46:12)



## Conclusion

- **Rollback recovery achieves fault tolerance** by periodically saving the state of a process during the failure-free execution, and restarting from a saved state on a failure to reduce the amount of lost computation.
- There are three basic approaches for checkpointing and failure recovery: **(i) uncoordinated, (ii) coordinated, and (iii) communication induced checkpointing** and for **log based: (i) Pessimistic, (ii) Optimistic and (iii) Casual Logging**
- Over the last two decades, **checkpointing and failure recovery** has been a very active area of research and several checkpointing and failure recovery algorithms have been proposed. In this lecture, we described **'Koo-Toueg Coordinated Checkpointing Algorithm'** and given an overview of other algorithms.

Cloud Computing and Distributed Systems | Failures & Recovery Approaches

Conclusion rollback recovery, achieves the fault tolerance. There are three different approaches: Uncoordinated, coordinated and communication induced and for log based pessimistic, optimistic and casual. Over the two decades checkpointing and failure recovery has been an active research and several algorithms for rollback have proposed. We have discussed one prominent algorithm in this direction Koo-Toueg coordinated checkpointing algorithm

Thank you.