Fault Tolerance

Part I Introduction

Part II Process Resilience

Part III Reliable Communication

Part IV Distributed Commit

Part V Recovery

Fault Tolerance Part I Introduction Part II Process Resilience

Presentation on theme: "Fault Tolerance Part I Introduction Part II Process Resilience"— Presentation transcript:

1 **Fault Tolerance Part I Introduction Part II Process Resilience**

Part III Reliable Communication

Part IV Distributed Commit

Part V Recovery

2 Fault Tolerance

Part I

Introduction

3 Faults

A system fails when it cannot meet its promises (specifications)

An error is part of a system state that may lead to a failure

A fault is the cause of the error
Fault-Tolerance: the system can provide services even in the presence of faults
Faults can be:
Transient (appear once and disappear)
Intermittent (appear-disappear-reappear behavior)
A loose contact on a connector intermittent fault
Permanent (appear and persist until repaired)

4 **Fault Tolerance A DS should be fault-tolerant**

Should be able to continue functioning in the presence of faults
Fault tolerance is related to dependability

5 **Dependability Dependability Includes Availability Reliability Safety**

Maintainability

6 **Availability & Reliability (1)**

Availability: A measurement of whether a system is ready to be used immediately
System is available at any given moment
Reliability: A measurement of whether a system can run continuously without failure
System continues to function for a long period of time

7 **Availability & Reliability (2)**

A system goes down 1ms/hr has an availability of more than 99.99%, but is unreliable
A system that never crashes but is shut down for a week once every year is 100% reliable but only 98% available

8 **Safety & Maintainability**

Safety: A measurement of how safe failures are
System fails, nothing serious happens
For instance, high degree of safety is required for systems controlling nuclear power plants
Maintainability: A measurement of how easy it is to repair a system
A highly maintainable system may also show a high degree of availability
Failures can be detected and repaired automatically? Self-healing systems?

9 **System reliability: Fault-Intolerance vs. Fault-Tolerance**

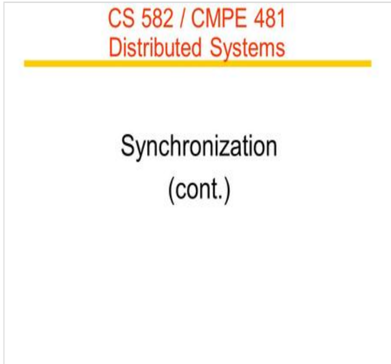
The fault intolerance (or fault-avoidance) approach improves system reliability by removing the source of failures (i.e., hardware and software faults) before normal operation begins
The approach of fault-tolerance expect faults to be present during system operation, but employs design techniques which insure the continued correct execution of the computing process
CS-550 (M.Soneru): Fault-tolerance [SaS]

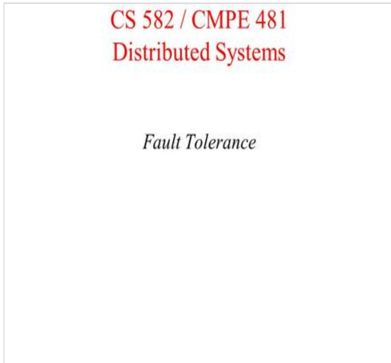
10 **Failure Models Type of failure Description Crash failure**

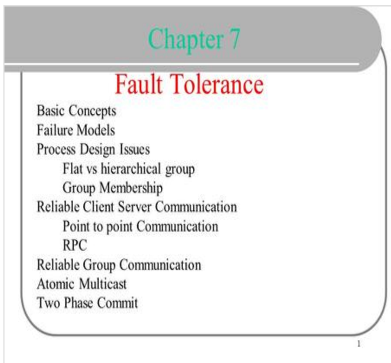
A server halts, but is working correctly until it halts
Omission failure Receive omission Send omission
A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure
A server's response lies outside the specified time interval
Response failure Value failure State transition failure
The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure
(Byzantine failure)
A server may produce arbitrary responses at arbitrary times

11 **CS-550 (M.Soneru): Fault-tolerance [SaS]**

Issues
Process Deaths:
All resources allocated to a process must be recovered when a process dies
Kernel and remaining processes can notify other cooperating processes
Client-server systems: client (server) process needs to be informed that the corresponding server (client) process died
Machine failure:
All processes running on that machine will die







Client-server systems: difficult to distinguish between a process and machine failure
Issue: detection by processes of other machines
Network Failure:
Network may be partitioned into subnets
Machines from different subnets cannot communicate
Difficult for a process to distinguish between a machine and a communication link failure
CS-550 (M.Soneru): Fault-tolerance [SaS]

12 Approaches to fault-tolerance

(a) Mask failures
(b) Well defined failure behavior
Mask failures:
System continues to provide its specified function(s) in the presence of failures
Example: voting protocols
Well defined failure behaviour:
System exhibits a well define behaviour in the presence of failures
It may or it may not perform its specified function(s), but facilitates actions suitable for fault recovery
Example: commit protocols
A transaction made to a database is made visible only if successful and it commits
If it fails, transaction is undone
Redundancy:
Method for achieving fault tolerance (multiple copies of hardware, processes, data, etc...)
CS-550 (M.Soneru): Fault-tolerance [SaS]

13 Failure Masking Redundancy is key technique for hiding failures

Redundancy types:
Information: add extra (control) information
Error-correction codes in messages
Time: perform an action persistently until it succeeds:
Transactions
Physical: add extra components (S/W & H/W)
Process replication, electronic circuits

14 CS-550 (M.Soneru): Fault-tolerance [SaS]

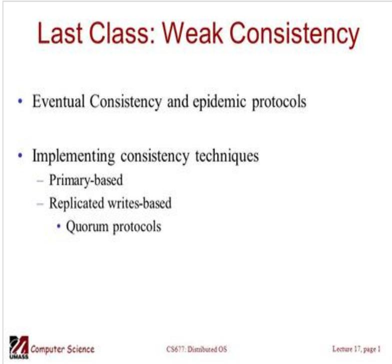
Voting protocols
Principles:
Data replicated at several sites to increase reliability
Each replica assigned a number of votes
To access a replica, a process must collect a majority of votes
Vote mechanism:
(1) Static voting:
Each replica has number of votes (in stable storage)
A process can access a replica for a read or write operation if it can collect a certain number of votes (read or write quorum)
(2) Dynamic voting
Number of votes or the set of sites that form a quorum change with the state of system (due to site and communication failures)
(2.1) Majority based approach:
Set of sites that can form a majority to allow access to replicated data of changes with the changing state of the system
(2.2) Dynamic vote reassignment:
Number of votes assigned to a site changes dynamically
CS-550 (M.Soneru): Fault-tolerance [SaS]

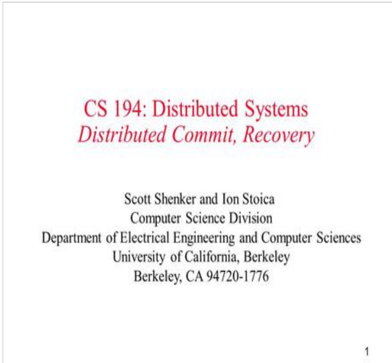
15 Fault Tolerance

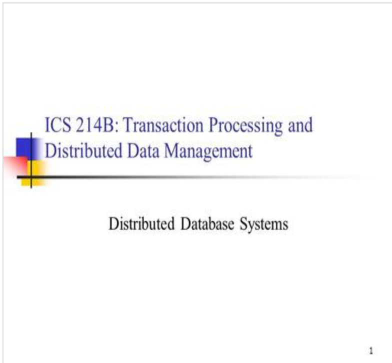
Part II
Process Resilience

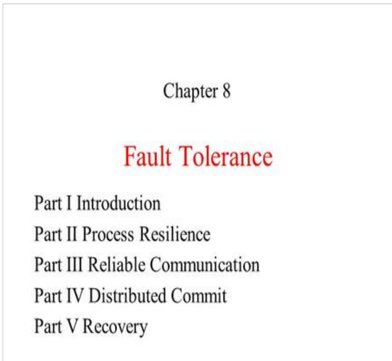
16 Failure resilient processes

Resilient process: continues execution in the presence of failures with minimum disruption to the service provided (masks failures)
Approaches for implementing resilient processes:
Backup processes and
Replicated execution
(1) Backup processes









Each process made of a primary process and one or more backup processes
Primary process execute, while the backup processes are inactive
If primary process fails, a backup process takes over
Primary process establishes checkpoints, such that backup process can restart
(2) Replicated execution
Several processes execute same program concurrently
Majority consensus (voting) of their results
Increases both the reliability and availability of the process
CS-550 (M.Soneru): Fault-tolerance [SaS]

17 **Process Resilience Mask process failures by replication**
Organize process into groups, a message sent to a group is delivered to all members
If a member fails, another should fill in

18 **Flat Groups versus Hierarchical Groups**
Communication in a flat group.
Communication in a simple hierarchical group

19 Process Replication
Replicate a process and group replicas in one group
How many replicas do we create?
A system is k fault-tolerant if it can survive and function even if it has k faulty processes
For crash failures
k+1 replicas
For Byzantine failures
2k+1 replicas

20 **Agreement Need agreement in DS:**
Leader, commit, synchronize
Distributed Agreement algorithm: all non-faulty processes achieve consensus in a finite number of steps
Perfect processes, faulty channels: two-army
Faulty processes, perfect channels: Byzantine generals

21 Two-Army Problem
In this example, Enemy Red Army has 5000 troops. Blue Army has two separate gatherings, Blue (1) and Blue (2), each of 3000 troops. Alone Blue will loose, together as a coordinated attack Blue can win. Communications is by unreliable channel (send a messenger who may be captured by red army so may not arrive.

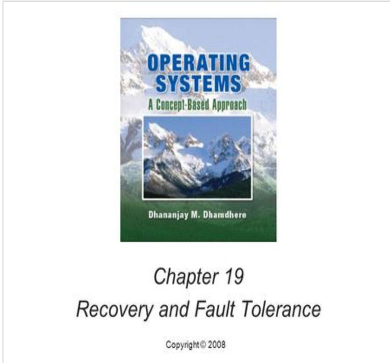
22 Two-Army Problem
In this example, Enemy Red Army has 5000 troops. Blue Army has two separate gatherings, Blue (1) and Blue (2), each of 3000 troops. Alone Blue will loose, together as a coordinated attack Blue can win. Communications is by unreliable channel (send a messenger who may be captured by red army so may not arrive.

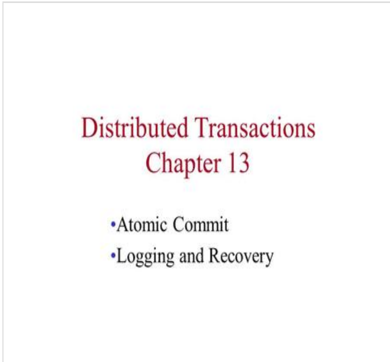
23 Impossible Consensus
Agreement is impossible in asynchronous DS, even if only one process fails [Fischer et al.]
Asynchronous DS:
messages cannot be guaranteed to be delivered within a known, finite time
cannot distinguish a slow process from a crashed one

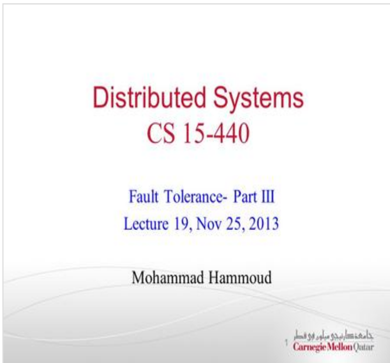
24 Possible Consensus
Agreement is possible in synchronous DS [e.g., Lamport et al.]
Messages can be guaranteed to be delivered within a known, finite time.
Byzantine Generals Problem
A synchronous DS: can distinguish a slow process from a crashed one

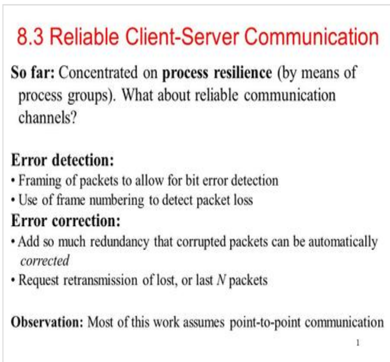
25 **Byzantine Generals Problem**

26 **Byzantine Generals -Example (1)**
The Byzantine generals problem for 3 loyal generals and1 traitor.
The generals announce the time to launch the attack (by messages marked by their ids).
The vectors that each general assembles based on (a)









The vectors that each general receives in step 3, where every general passes his vector from (b) to every other general.

27 **Byzantine Generals –Example (2)**

The same as in previous slide, except now with 2 loyal generals and one traitor.

28 Byzantine Generals

Given three processes, if one fails, consensus is impossible

Given N processes, if F processes fail, consensus is impossible if N 3F

29 **Reliable Communication**

Fault Tolerance

Part III

Reliable Communication

30 **Reliable Group Communication**

31 **Reliable Group Communication**

When a group is static and processes do not fail

Reliable communication = deliver the message to all group members

Any order delivery

Ordered delivery

32 **Basic Reliable-Multicasting Schemes**

A simple solution to reliable multicasting when all receivers are known and are assumed not to fail

Message transmission

Reporting feedback

33 Atomic Multicast

All messages are delivered in the same order to “all” processes

Group view: the view on the set of processes contained in the group

Virtual synchronous multicast: a message m multicast to a group view G is delivered to all non-faulty processes in G

If sender fails “before” m reaches a non-faulty process, none of the processes deliver m

34 **Virtual Synchrony System Model**

The logical organization of a distributed system to distinguish between message receipt and message delivery

35 **Reliability of Group Communication?**

A sent message is received by all members

(acks from all => ok)

Problem: during a multicast operation

an old member disappears from the group

a new member joins the group

Solution

membership changes synchronize multicasting

during a MC operation no membership changes

Virtual synchrony: “all” processes see message and membership change in the same order

36 Part IV

Distributed commit

37 Distributed Commit

Goal: Either all members of a group decide to perform an operation, or none of them perform the operation

Atomic transaction: a transaction that happens completely or not at all

Is required in distributed transactions

38 **Assumptions Failures: Notes: Crash failures that can be recovered**

Communication failures detectable by timeouts

Notes:

Commit requires a set of processes to agree…

…similar to the Byzantine generals problem…

... but the solution much simpler because stronger assumptions

39 Distributed Transactions

Database
ser-ver
client
Database
ser-ver
atomic
Atomic
Consistent
Isolated
Durable
isolated
serializable
client
ser-ver

40 A Distributed Banking Transaction

openTransaction
join
participant
closeTransaction
A
a.withdraw(4);
. .
join
BranchX
T
participant
Client
B
b.withdraw(3);
T =
openTransaction
BranchY
a.withdraw(4);
join
c.deposit(4);
participant
b.withdraw(3);
d.deposit(3);
c.deposit(4);
C
closeTransaction
D
d.deposit(3);
BranchZ

41 Distributed commit Is done by using atomic commitment protocols (ACP)

One-phase Commit
Two-phase Commit
Three-phaseCommit

42 One-phase Commit One-phase commit protocol

One site is designated as a coordinator
The coordinator tells all the other processes whether or not to locally perform the operation in question
This scheme however is not fault tolerant

43 Two Phase Commit (2PC) Coordinator Participants send VOTE_REQ to all

send vote to coordinator
if (vote == no)
decide abort
halt
if (all votes yes)

decide commit
send COMMIT to all
else
decide abort
send ABORT to all who voted yes
halt
if receive ABORT, decide abort
else decide commit
halt

44 Two-Phase Commit (1) X

The finite state machine for the coordinator in 2PC.
The finite state machine for a participant.

45 Two-Phase Commit (2)

State of Q
Action by P
COMMIT
Make transition to COMMIT
ABORT
Make transition to ABORT
INIT
READY
Contact another participant
Actions taken by a participant P when residing in state READY and having contacted another participant Q.

46 Two-Phase Commit(3)

When all participants are in the ready states, no final decision can be reached
Two-phase commit is a blocking commit protocol

47 Three-Phase Commit (1)

There is no state from which a transition can be made to either Commit or Abort
There is no state where it is not possible to make a final decision and from which transition can be made to Commit
non-blocking commit protocol

48 **Three-Phase Commit (2) Coordinator sends Vote_Request (as before)**

If all participants respond affirmatively,
Put Precommit state into log on stable storage
Send out Prepare_to_Commit message to all
After all participants acknowledge,
Put Commit state in log
Send out Global_Commit

49 **Three-Phase Commit (3) Coordinator blocked in Wait state**

Safe to abort transaction
Coordinator blocked in Precommit state
Safe to issue Global_Commit
Any crashed or partitioned participants will commit when recovered
…

50 **Three-Phase Commit (4) Participant blocked in Precommit state**

Contact others
Collectively decide to commit
Participant blocked in Ready state
If any in Abort, then abort transaction
If any in Precommit, the move to Precommit state
If all in Ready state, then abort transaction

51 Fault Tolerance

Part V
Recovery

52 Recovery

We've talked a lot about fault tolerance, but not about what happens after a fault has occurred
A process that exhibits a failure has to be able to recover to a correct state
There are two basic types of recovery:
Backward Recovery
Forward Recovery

53 Backward Recovery

The goal of backward recovery is to bring the system from an erroneous state back to a prior correct state
The state of the system must be recorded - checkpointed - from time to time, and then restored when things go wrong
Examples
Reliable communication through packet retransmission

54 Forward Recovery

The goal of forward recovery is to bring a system from an erroneous state to a correct new state (not a previous state)
Examples:
Reliable communication via erasure correction, such as an (n, k) block erasure code

55 **More on Backward Recovery**

Backward recovery is far more widely applied
Checkpointing is costly, so it's often combined with message logging

56 Stable Storage

In order to store checkpoints and logs, information needs to be stored safely - not just able to survive crashes, but also able to survive hardware faults
RAID (of the mirroring or parity checking variety) is the typical example of stable storage

57 Checkpointing

Related to checkpointing, let us first discuss the global state and the distributed snapshot algorithm

58 **Determining Global States**

The Global State of a distributed computation is
the set of local states of all individual processes involved in the computation
+
the states of the communication channels
How?

59 **Obvious First Solution...**

Synchronize clocks of all processes and ask all processes to record their states at known time t
Problems?
Time synchronization possible only approximately
distributed banking applications: no approximations!
Does not record the state of messages in the channels

60 Global State

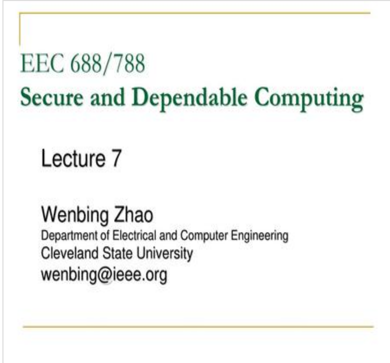
We cannot determine the exact global state of the system, but we can record a snapshot of it
Distributed Snapshot: a state the system might have been in [Chandy and Lamport]

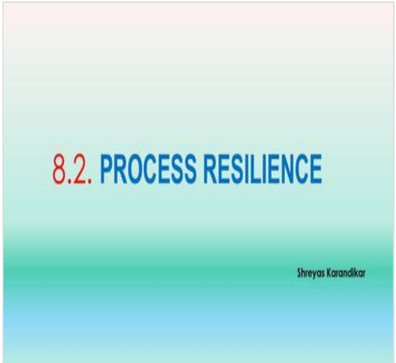
61 **A naïve snapshot algorithm**

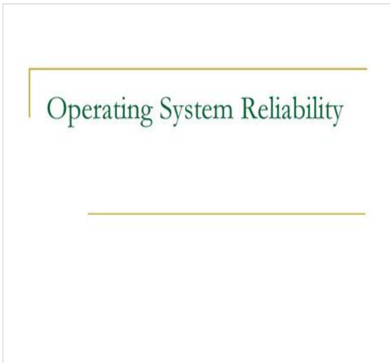
Processes record their state at any arbitrary point
A designated process collects these states
+ So simple!!
- Correct??

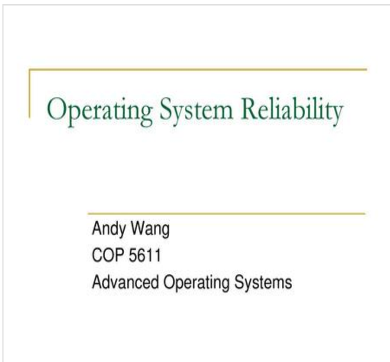
62 **The "Snapshot" Algorithm**

Records a set of process and channel states such that the combination is a consistent GS.
Assumptions:
No failure, all messages arrive intact, exactly once









Communication channels are unidirectional and FIFO-ordered
There is a comm. path between any two processes
Any process may initiate the snapshot (sends Marker)
Snapshot does not interfere with normal execution
Each process records its state and the state of its incoming channels

63 **The "Snapshot" Algorithm (2)**

1. Marker sending rule for initiator process P0
After P0 has recorded its state
for each outgoing channel C, sends a marker on C
2. Marker receiving rule for a process Pk, on receipt of a marker over channel C
if Pk has not yet recorded its state
records Pk's state
records the state of C as "empty"
turns on recording of messages over other incoming channels
else
records the state of C as all the messages received over C since Pk saved its state

64 **Snapshot Example P1 P2 P3 e14 e11,2 e10 e13 e21,2,3 e31,2,3 e25 e24**

3- P1 receives Marker over C21, sets state(C21) = {a}
M
e11,2
1- P1 initiates snapshot: records its state (S1); sends Markers to P2 & P3; turns on recording for channels C21 and C31
e10
e13
P1
e21,2,3
M
2- P2 receives Marker over C12, records its state (S2), sets state(C12) = {} sends Marker to P1 & P3; turns on recording for channel C32
e31,2,3
M
4- P3 receives Marker over C13, records its state (S3), sets state(C13) = {} sends Marker to P1 & P2; turns on recording for channel C23
e25
5- P2 receives Marker over C32, sets state(C32) = {b}
a
e24
P2
e20
b P3
e34
6- P3 receives Marker over C23, sets state(C23) = {}
e30

65 Snapshot Example

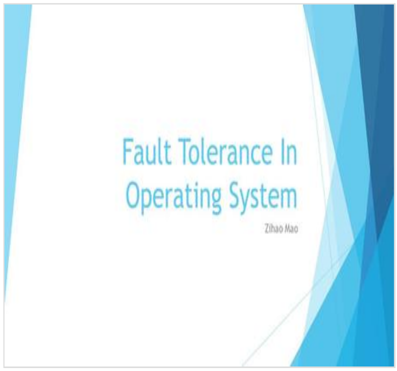
e10
e13
P1 a
e24
P2
e20
b P3
e30

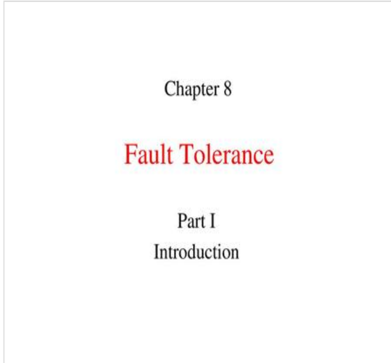
66 **Distributed Snapshot Algorithm**

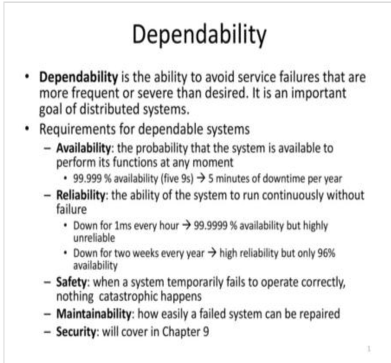
When a process finishes local snapshot, it collects its local state (S and C) and sends it to the initiator of the distributed snapshot
The initiator can then analyze the state
One algorithm for distributed global snapshots, but it's not particularly efficient for large systems

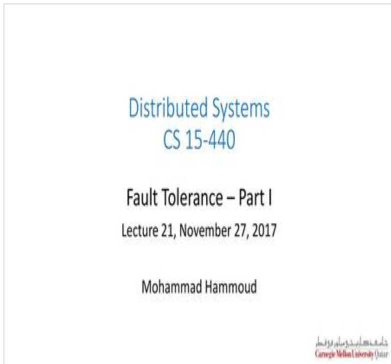
67 **Checkpointing We've discussed distributed snapshots**

The most recent distributed snapshot in a system is also called the recovery line









68 **Independent Checkpointing**

It is often difficult to find a recovery line in a system where every process just records its local state every so often - a domino effect or cascading rollback can result:

69 **Coordinated Checkpointing**

To solve this problem, systems can implement coordinated checkpointing
We've discussed one algorithm for distributed global snapshots, but it's not particularly efficient for large systems
Another way to do it is to use a two-phase blocking protocol (with some coordinator) to get every process to checkpoint its local state "simultaneously"

70 **Coordinated Checkpointing**

Make sure that processes are synchronized when doing the checkpoint
Two-phase blocking protocol
Coordinator multicasts CHECKPOINT_REQUEST
Processes take local checkpoint
Delay further sends
Acknowledge to coordinator
Send state
Coordinator multicasts CHECKPOINT_DONE

71 Message Logging

Checkpointing is expensive - message logging allows the occurrences between checkpoints to be replayed, so that checkpoints don't need to happen as frequently

72 **Message Logging We need to choose when to log messages**

Message-logging schemes can be characterized as pessimistic or optimistic by how they deal with orphan processes
An orphan process is one that survives the crash of another process but has an inconsistent state after the other process recovers

73 Message Logging

An example of an incorrect replay of messages

74 Message Logging

We assume that each message m has a header containing all the information necessary to retransmit m (sender, receiver, sequence no., etc.)
A message is called stable if it can no longer be lost - a stable message can be used for recovery by replaying its transmission

75 Message Logging

Each message m leads to a set of dependent processes DEP(m), to which either m or a message causally dependent on m has been delivered

76 Message Logging

The set COPY(m) consists of the processes that have a copy of m, but not in their local stable storage - any process in COPY(m) could deliver a copy of m on request

77 Message Logging

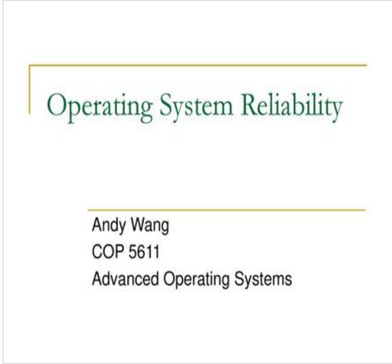
Process Q is an orphan process if there is a nonstable message m, such that Q is contained in DEP(m), and every process in COPY(m) has crashed

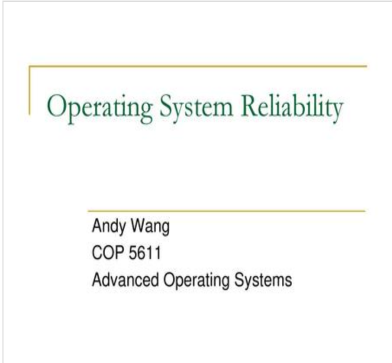
78 Message Logging

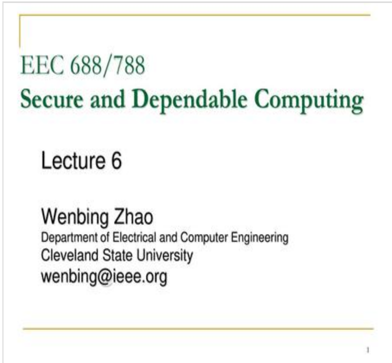
To avoid orphan processes, we need to ensure that if all processes in COPY(m) crash, no processes remain in DEP(m)

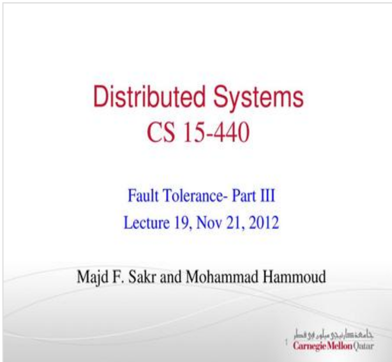
79 Pessimistic Logging

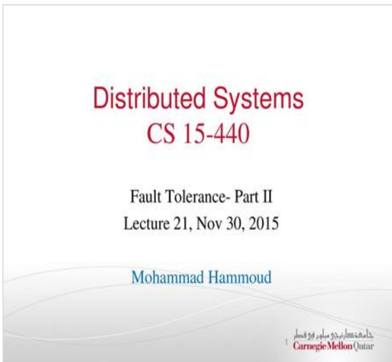
For each nonstable message m, ensure that at most one process P is dependent on m
The worst that can happen is that P crashes without m ever having been logged
No other process can have become dependent on m, because m was nonstable, so this leaves no orphans











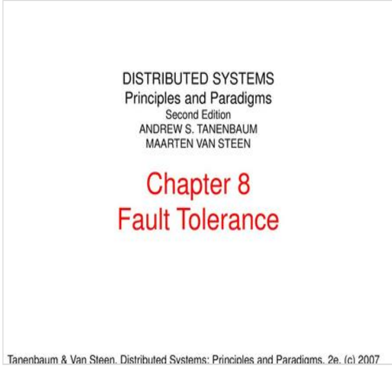
80 **Optimistic Logging The work is done after a crash occurs, not before**

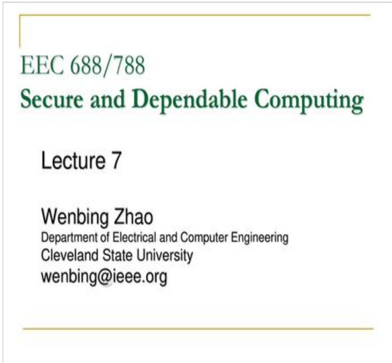
If, for some m, each process in COPY(m) has crashed, then any orphan process in DEP(m) gets rolled back to a state in which it no longer belongs in DEP(m)

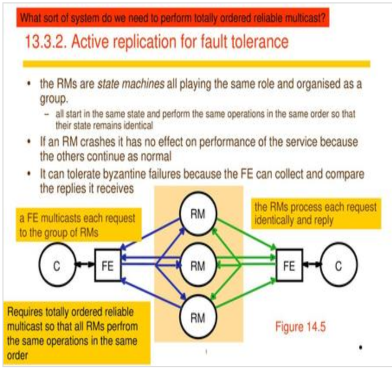
81 **Optimistic Logging The work is done after a crash occurs, not before**

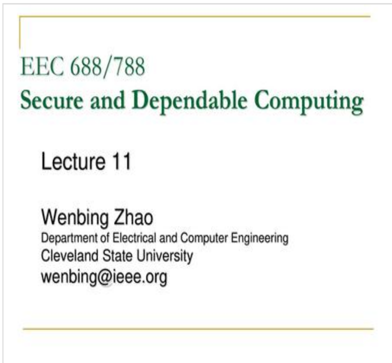
If, for some m, each process in COPY(m) has crashed, then any orphan process in DEP(m) gets rolled back to a state in which it no longer belongs in DEP(m)
Dependencies need to be explicitly tracked, which makes this difficult to implement - as a result, pessimistic approaches are preferred in real-world implementations

---









CSE 486/586 Distributed Systems
Concurrency Control --- 3

Steve Ko
Computer Sciences and Engineering
University at Buffalo

CSE 486/586

EEC 688/788
Secure and Dependable Computing

Lecture 11

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

EEC 688/788
Secure and Dependable Computing

Lecture 7

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

Distributed Systems
CS 15-440

Fault Tolerance
Lecture 25, December 06, 2018

Mohammad Hammoud

Operating System Reliability

Andy Wang
COP 5611
Advanced Operating Systems

EEC 688/788
Secure and Dependable Computing

Lecture 6

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

EEC 688/788
Secure and Dependable Computing

Lecture 11

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

EEC 688/788
Secure and Dependable Computing

Lecture 10

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

EEC 688/788
Secure and Dependable Computing

Lecture 10

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

EEC 688/788
Secure and Dependable Computing

Lecture 11

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

EEC 688/788
Secure and Dependable Computing

Lecture 11

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

Fault Tolerance and Reliability in DS

EEC 688/788
Secure and Dependable Computing

Lecture 6

Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University
wenbing@ieee.org

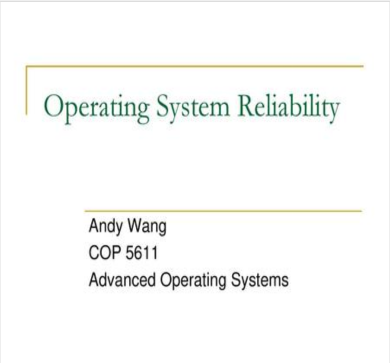
Abstractions for Fault Tolerance

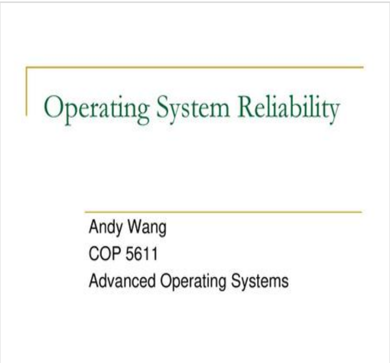
Kris Malfettone, Adrian Dumchus

CSE 486/586 Distributed Systems
Concurrency Control --- 3

Steve Ko
Computer Sciences and Engineering
University at Buffalo


Last Class: Fault Tolerance
• Basic concepts and failure models
• Failure masking using redundancy
• Agreement in presence of faults
– Two army problem
– Byzantine generals problem
Computer Science


Operating System Reliability
Andy Wang
COP 5611
Advanced Operating Systems


Operating System Reliability
Andy Wang
COP 5611
Advanced Operating Systems