

EEC 693/793
Special Topics in Electrical Engineering
Secure and Dependable Computing

Lecture 11

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

2

Outline

- **Midterm #2 (lecture 7-11, lab 1-3)**
 - **April 3, Monday, 6-8pm**
- Replication and consistency concepts
- Replication techniques
- Handling replica non-determinisms
- Two-phase commit

2 April 2006 EEC693/793 Wenbing Zhao

3

Redundancy

- To tolerant fault, some form of redundancy must be used
- Replication in time (transaction processing):
 - From the present state, apply one or more operations
 - If the system fails before completion, abort and rollback to the original state
 - Start all over again
- Replication in space:
 - Run multiple instances (replicas) of the systems so that if one fails, one or more replicas can take over
 - Must assume independent faults
- N-version programming (redundancy in software design)
 - The system (or component) has n-different design and implementations
 - In case of permanent software bugs, a different version is used for each replica, or for repeated executions

2 April 2006 EEC693/793 Wenbing Zhao

4

Replication is not a Trivial Task

- Suppose we want to replicate a server using the most popular (an inexpensive) approach
 - We run two servers on separate computers
 - The primary sends a log (its state, and/or logged incoming messages) to the backup
 - If primary crashes, the backup soon catches up and can take over

2 April 2006 EEC693/793 Wenbing Zhao

Split Brain Syndrome

5

Clients initially connected to primary, which keeps backup up to date. Backup collects the log and update its state

2 April 2006 EEC693/793 Wenbing Zhao

Split Brain Syndrome

6

Transient problem causes some links to break but not all. Backup thinks it is now primary, primary thinks backup is down

2 April 2006 EEC693/793 Wenbing Zhao

Split Brain Syndrome

7

Some clients still connected to primary, but one has switched to backup and one is completely disconnected from both

2 April 2006 EEC693/793 Wenbing Zhao

Consistency

8

- Replicas must be coordinated appropriately so that we can achieve strong replica consistency
 - At the end of each execution step, all replicas must have the same state
 - The outputs from every replica must be consistent all the time
- Techniques to achieve strong replica consistency
 - As a prerequisite, we need to have an agreement on the **membership** of the replica group
 - Ensure that the same set of inputs is delivered to all replicas and the inputs must be in the same order
 - Ensure deterministic execution of each input at every replica (applicable to active and semi-active replication)

2 April 2006 EEC693/793 Wenbing Zhao

One Copy Image

- When a component or system is replicated, we must ensure that the replicated unit appears as a single copy to external components/systems that interact with it
- Need a gateway in between the replicated and non-replicated units
- Alternatively, the gateway module can be integrated with the non-replicated unit
- Primary tasks of the gateway
 - Multicast requests/replies to the replicas
 - Detect and suppress duplicated replies/requests

Replication Styles

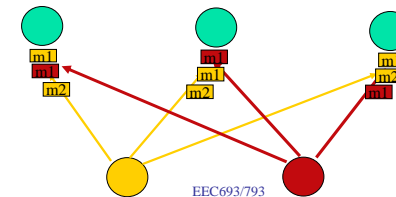
- Active replication
 - Every input (request) is executed by every replica
 - Every replica generates the outputs (replies)
 - Voting is needed to cope with non-fail-stop faults
- Passive replication
 - One of the replicas is designated as the primary replica
 - Only the primary replica executes requests
 - The state of the primary replica is transferred to the backups periodically or after every request processing
- Semi-active replication
 - One of the replicas is designated as the leader (or primary)
 - The leader determines the order of execution
 - Every input is executed by every replica per the leader's instruction

Ensuring Strong Replica Consistency

- strategies
 - For active replication, use a group communication system that guarantees total ordering of all messages (plus deterministic processing in each replica)
 - Passive replication with systematic checkpointing
 - Semi-active replication
 - Use two-phase commit

Total Ordering of Messages

- What is total ordering of messages?
 - All replicas receive the same set of messages in the same order
 - Atomic multicast – If a message is delivered to one replica, it is also delivered to all correct replicas
- With replication, we need to ensure total ordering of messages sent by a group of replicas to another group of replicas
 - FIFO ordering between one sender and a group is not sufficient



Potential Sources of Non-determinisms

- Multithreading
 - The order of accesses of shared data by different threads might not be the same at different replicas
- System calls/library calls
 - A call at one replica might succeed while the same call might fail at another replica
 - ◆ Memory allocation
 - ◆ File access
- Host/process specific information
 - Host name, process id, etc.
 - Local clocks - gettimeofday()
- Interrupts
 - Delivered and handled asynchronously – big problem

Replica Non-determinism Sanitization

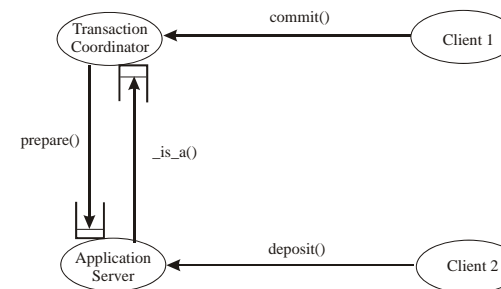
- Hypervisor approach – operates between hardware and operating system
 - Handle interrupts at the same point
 - ◆ Use an instruction counter
 - ◆ Need to choose a leader among the replicas
 - ◆ The leader makes a decision when to handle the interrupt and inform other replicas to do the same
- Other nondeterministic calls are emulated in software
 - The call is intercepted, the call is not returned until the coordination among the replicas is completed
 - Either use a leader to make decision
 - Or the replicas collectively make a decision

Replica Non-determinism Sanitization

- Handling of multithreaded replicas
 - Do not replica multithreaded application (state-machine approach – very popular assumption in research work)
 - Serialize multithreaded execution – many problems, not just serious performance degradation
 - Leader-based approach
 - Optimistic and semi-optimistic locking (ongoing research)

Replica Non-determinism Sanitization

- Serialization of a multithreaded application might lead to deadlock



Passive Replication with Systematic Checkpointing

17

- Before the sending of a reply to the client, the primary takes a checkpoint of its state and multicasts the reply together with the checkpoint to both the client and the other replicas (backups) atomically

2 April 2006

EEEC693/793

Wenbing Zhao

Semi-Active Replication

18

- Before the sending of a reply to the client, the leader piggyback the ordering information with the reply and multicasts the combined message to both the client and the other replicas atomically

2 April 2006

EEEC693/793

Wenbing Zhao

Coping with Replica Non-determinism

19

- Why the systematic checkpointing and semi-active replication can cope with replica non-determinism?
- If a replica does not reveal its state to external environment, such as clients, it won't cause any replica inconsistency problem
 - The new leader/primary, or another replica in active replication, can start from the previous visible state and continue on with its own decision

2 April 2006

EEEC693/793

Wenbing Zhao

Two-Phase Commit

20

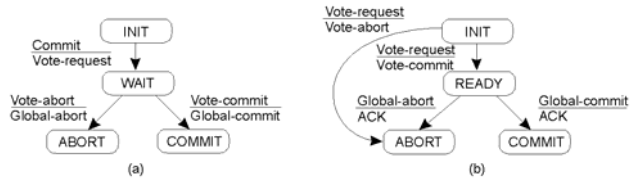
- **Model:** The client who initiated the computation acts as a coordinator; processes required to commit are the participants
- **Phase 1a:** Coordinator sends VOTE_REQUEST to participants (also called a **pre-write**)
- **Phase 1b:** When participant receives VOTE_REQUEST it returns either YES or NO to coordinator. If it sends NO, it aborts its local computation
- **Phase 2a:** Coordinator collects all votes; if all are YES, it sends COMMIT to all participants, otherwise it sends ABORT
- **Phase 2b:** Each participant waits for COMMIT or ABORT and handles accordingly

2 April 2006

EEEC693/793

Wenbing Zhao

Two-Phase Commit



The finite state machine for the coordinator in 2PC

The finite state machine for a participant

2PC – Failing Participant

- Consider participant crash in one of its states, and the subsequent recovery to that state:
- **Initial state:** No problem, as participant was unaware of the protocol
- **Ready state:** Participant is waiting to either commit or abort. After recovery, participant needs to know which state transition it should make => log the coordinator's decision
- **Abort state:** Need to make entry into abort state *idempotent*
- **Commit state:** Also make entry into commit state *idempotent*

2PC – Failing Coordinator

- If it fails, the final decision is not available until the coordinator recovers
- **Alternative:** Let a participant *P* in the ready state timeout when it hasn't received the coordinator's decision
 - *P* tries to find out what other participants know
- **Question:** Can *P* not succeed in getting the required information?
- **Observation:** Essence of the problem is that a recovering participant cannot make a **local** decision: it is dependent on other (possibly failed) processes
 - There might exist one participant that has received a COMMIT decision from the coordinator and subsequently failed (more or less concurrently failed with the coordinator)
 - The rest of participants cannot unilaterally decide to abort the transaction

Two-Phase Commit

- Why 2PC can help achieve replica consistency?
 - The processing of every input is carried out atomically
 - ❖ Either all replicas succeed, or none
 - ❖ E.g., one replica might not be able to complete the task due to deadlock avoidance (one form of nondeterminism). The whole operation will be aborted and retried
- 2PC is more appropriate with replicated applications using the read/write operations with structured data, such as replicated database systems