Highly Concurrent Fault Tolerance Computing using Software Transactional Memory

Wenbing Zhao

Cleveland State University
wenbing@ieee.org

Outline

- Motivation
- Background
- Fault tolerance computing using STM
- Conclusion

Motivation

- Replication is an essential technique to ensure high availability
- Active replication (state machine replication)
  - Requires deterministic replicas
  - Not appropriate for multithreaded applications
- Passive replication
  - Requires frequent incremental state transfer
  - Either resort to process state checkpointing (very inefficient) or imposing the duty on applications (intrusive and error prone)

Approach

- We decide to use passive replication with software transactional memory
- The combination of the two enables highly concurrent fault tolerance computing

Software Transactional Memory

- STM is a concurrency control mechanism for controlling access to shared memory
  - It is analogous to transaction processing: a group of instructions is executed atomically
  - It serves as an alternative to lock-based synchronization
  - STM is optimistic: all changes within a transaction are temporary until the transaction is committed. If the intermediate result is exposed to another thread, the transaction is aborted

Software Transactional Memory

- Plan to use an open-source STM library, XSTM, for this research
- XSTM:
  - Object based STM in Java
  - Provide preliminary object replication
  - Two-phase commit is used for replica coordination. However, the coordinator is assumed to be failure-free

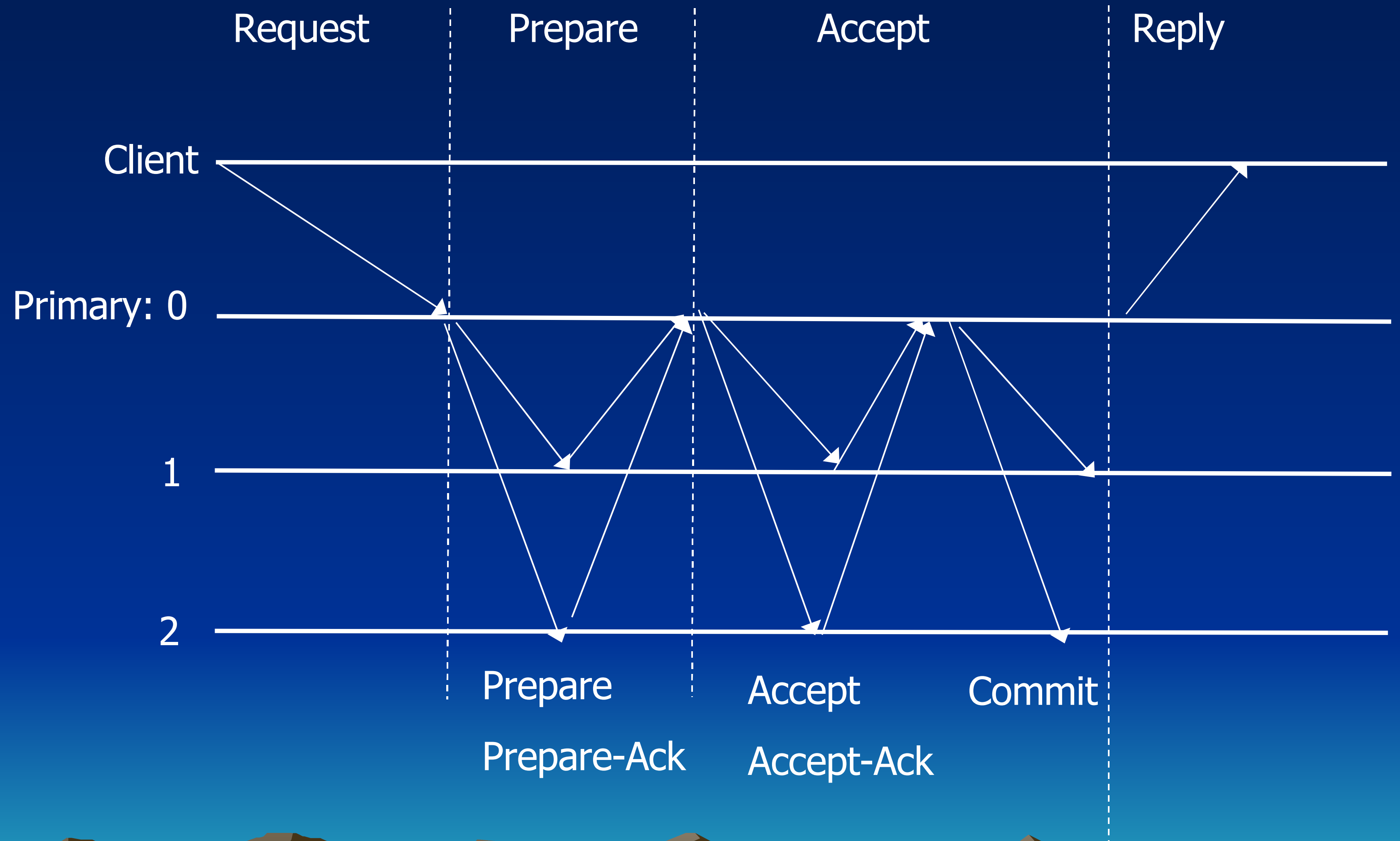
Fault Tolerance Computing Using STM

- System model:
  - Asynchronous systems with fail-stop faults
  - 2f+1 server replicas to tolerate up to f faulty replicas (one as primary and 2f as backups)
  - Each remote operation is mapped into a single transaction
  - Consider only client-server interactions

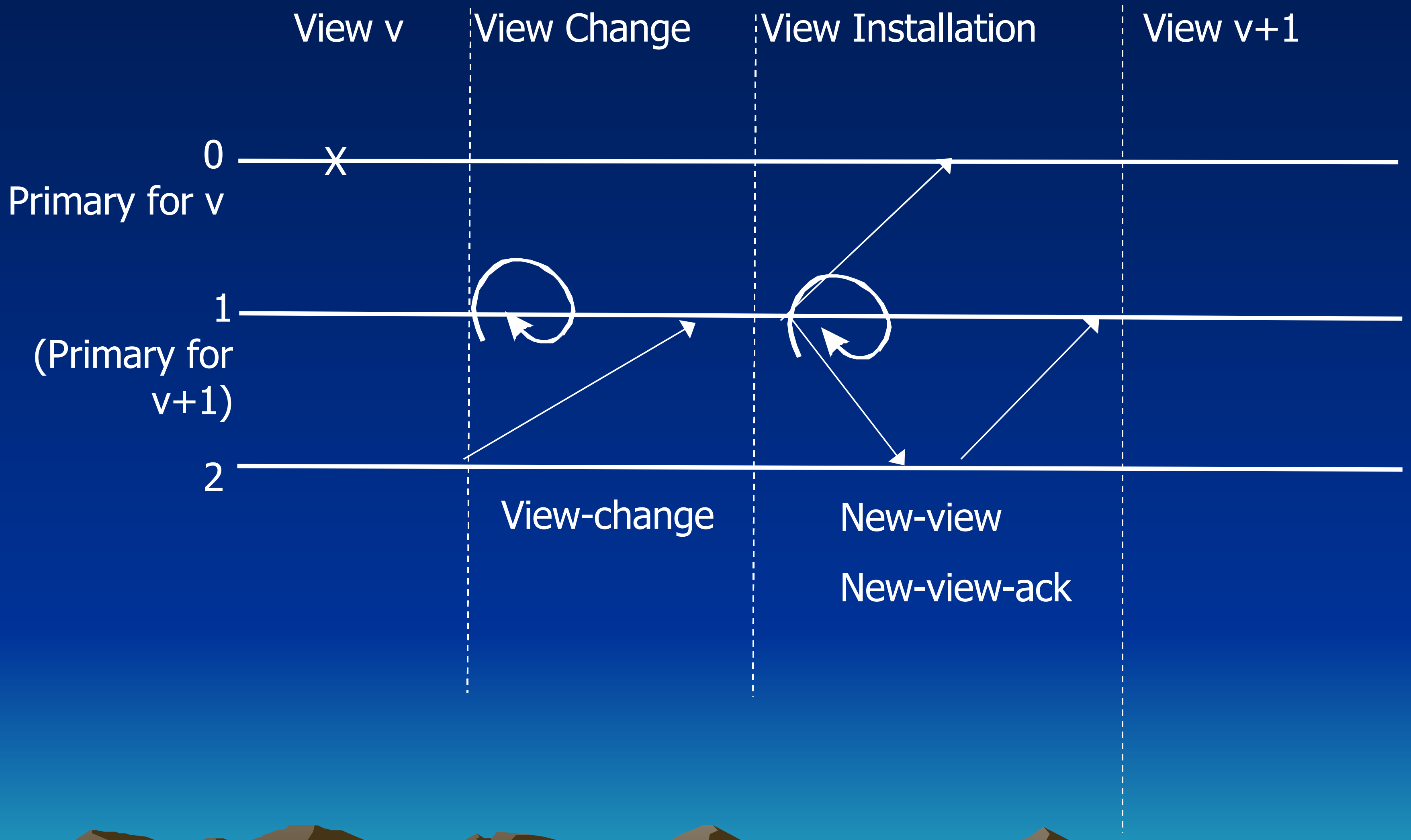
Fault Tolerance Computing Using STM

- Basic operation
  - Client sends its request to the primary, if it does not receive a response promptly, it retransmit to all server replicas
  - The primary executes the request and coordinate with backups using a replication algorithm before it ships the response out
  - Each backup maintains a timer to detect the primary failure. It initiates a view change (i.e., select a new primary) when timer expires

Replication Algorithm – Normal Operation



Replication Algorithm – View Change

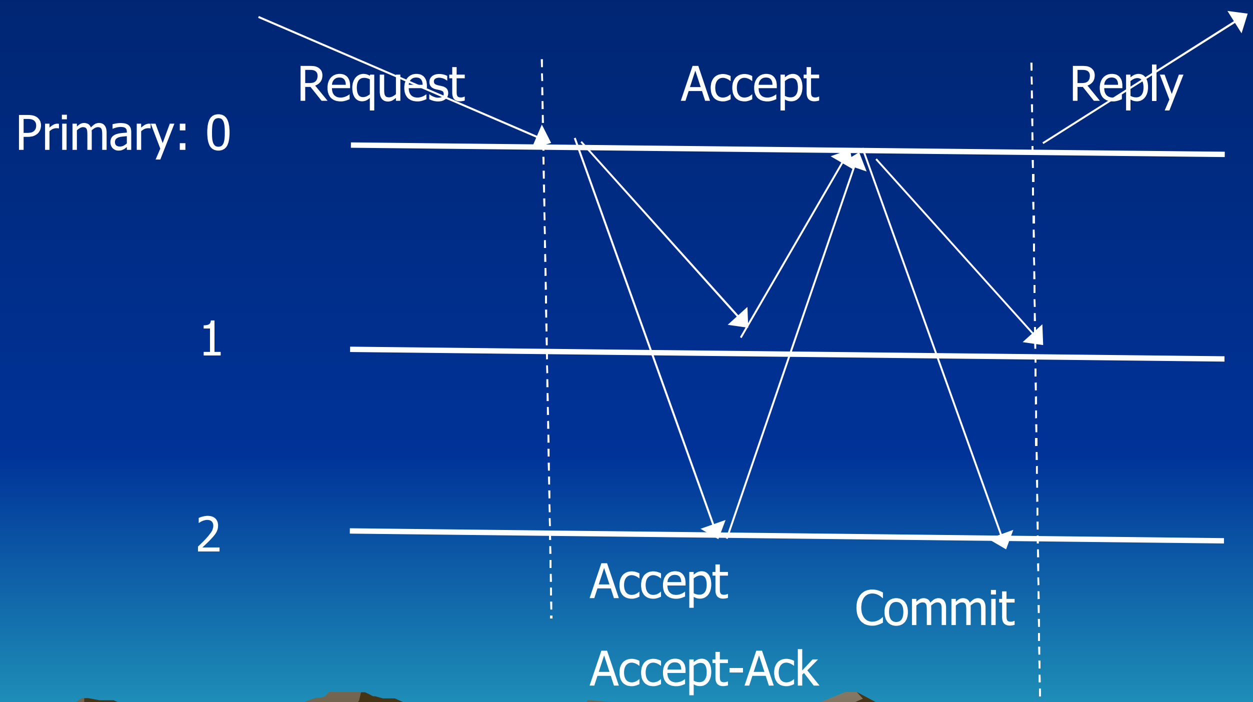


Replication Coordination - Details

- At the end of execution of a request, the primary tries to commit the transaction.
  - If it is aborted, the primary will reinsert the request in the queue for retry. If it is committed, the primary starts the replication algorithm for the request
- What information is included in the messages exchanged between the primary and backups?
  - Client's request
  - All objects that have been updated during the execution of the request
  - The response to the client's request

Optimization

- Assuming the initial membership of the replicas is determined statically, which is typically the case, the prepare phase can be omitted



Optimization

- Load balancing: partitioning state, each partition is in charge by one replica and that replica is the primary for its partition
  - All replicas are actively executing, further increasing the throughput
  - If a transaction spans more than one partition, a distributed commit will have to take place => requires more sophisticated infrastructure support (i.e., a replicated coordination service)

Conclusion and Future Work

- We propose to use passive replication and STM to enable highly concurrent fault tolerance computing
- Future work
  - Implementation of the proposed framework
  - Identify practical applications and demonstrate the effectiveness of our framework