Non-blocking Atomic Commitment

Aaron Kaminsky
Agenda

- Atomic Commitment Problem
- Model and Terminology
- One-Phase Commit (1PC)
- Generic Atomic Commit Protocol (ACP)
- Two-Phase Commit (2PC)
- Non-Blocking ACP
- Three-Phase Commit (3PC)
Atomic Commitment

- Distributed transaction involves different processes operating on local data
- Partial failure can result in an inconsistent state
- Atomic commitment - either all processes commit or all abort
System Model

- Distributed system using messages for communication
- Synchronous model
  - Bounds exist and are known for process speeds
  - Bounds exist and are known for potential message delays
Communication

- Assume reliable communication
- Assume defined upper bound for processing and transmission delays
- $\delta =$ time units between send and receive (includes processing at sender and receiver)
- Timeouts can be used to detect process failure
Process

- Operational – executes the program
- Down – performs no action
- Crash – move from operational to down
- Correct – has never crashed
- Faulty – has crashed
Distributed Transactions

- Each participant updates local data
- The invoker begins the transaction by sending a message to all participants
  - Piece of the transaction
  - List of participants
  - $\Delta_c$ – time until the transaction should be concluded
Distributed Transactions Cont.

- Each process sets a local variable, vote at the end of processing
  - vote = YES – local operation successful, results can be made permanent
  - vote = NO – some failure prevents updating local data with results
- Finally the Atomic Commitment Protocol is used to decide the outcome of the transaction
The Atomic Commitment

Problem

- **AC1**: all participants that decide reach the same decision.
- **AC2**: if any participant decides *commit*, then all participants must have voted YES.
- **AC3**: if all participants vote YES and no failures occur, then all participants decide *commit*.
- **AC4**: each participant decides at most once (that is, a decision is irreversible).
One-Phase Commit Protocol

- Elect a coordinator
- Coordinator tells all participants whether or not to locally commit results
- Cannot handle the failure of a participant
1PC In Action

Coordinator

P₁

P₂

COMMIT

COMMIT
Generic Atomic Commitment Protocol (ACP)

- Modification to 2PC
- Broadcast algorithm is left undefined
- $C_{\text{know}} = \text{Local time when participant learns of the transaction}$
- $\Delta_c = \text{upper bound for time from } C_{\text{know}} \text{ to coordinator concluding transaction}$
- $\Delta_b = \text{upper bound for time from broadcast of message to delivery of message}$
ACP Coordinator Algorithm

send [VOTE_REQUEST] to all participants
set timeout to local_clock + 2δ
wait for [vote:vote] from all participants
if all votes = YES then
    broadcast commit to all participants
else broadcast abort to all participants
on timeout broadcast abort to all participants
ACP Participant Algorithm

set timeout to \((C_{\text{know}} + \Delta_c + \delta)\)

wait for \([\text{VOTE\_REQUEST}]\) from the coordinator

send \([\text{vote}: \text{vote}]\) to the coordinator

if \((\text{vote} = \text{NO})\) decide(ABORT)

else

set timeout to \((C_{\text{know}} + \Delta_c + \delta + \Delta_b)\)

wait for delivery of decision message

if \((\text{decision} = \text{abort})\) decide(\text{abort})

else decide(\text{commit})

on timeout decide according to termination protocol

on timeout decide(\text{abort})
SB1: A Simple Broadcast Algorithm

// broadcaster executes:
send [DLV: m] to all processes in G
deliver m

// process p <> broadcaster in G executes
upon (receipt of [DLV: m])
deliver m
Properties of SB1

- **B1 (Validity):** If a correct process broadcasts a message \( m \), then all correct processes in \( G \) eventually deliver \( m \).

- **B2 (Integrity):** For any message \( m \), each process in \( G \) delivers \( m \) at most once, and only if some process actually broadcasts \( m \).

- **B3 (\( \Delta_b \)-Timeliness):** There exists a known constant \( \Delta_b \) such that if the broadcast of \( m \) is initiated at real-time \( t \), no process in \( G \) delivers \( m \) after real-time \( t + \Delta_b \).
Combine to get ACP-SB

- This is equivalent to 2PC in the Tanenbaum text
- The paper proves that this protocol solves the Atomic Commitment Problem as defined earlier.
Coordinator initiates vote by sending VOTE_REQUEST to participants
Coordinator receives response from participants
ACP-SB In Action

Coordinator broadcasts decision to participants

Coordinator

ABORT

ABORT

P₁

P₂
Blocking

- ACP-SB1 can result in blocking when the coordinator goes down
- Traditional solution - poll peers to determine decision
- It can still happen that participants must block and wait for the coordinator to recover
- Resources are not released
Coordinator receives all YES votes
Coordinator and $P_2$ go down, $P_1$ never gets COMMIT

$P_1$ must block until Coordinator recovers
The Non-Blocking Atomic Commitment Problem

- Now the goal is to prevent blocking
- Add a new requirement to the protocol
- AC5: every correct participant that executes the atomic commitment protocol eventually decides.
Uniform Timed Reliable Broadcast (UTRB)

- To B1-B3 (Validity, Integrity and $\Delta_b$-Timeliness) add another requirement.
- B4 (Uniform Agreement): If any process (correct or not) in G delivers a message $m$, then all correct processes in G eventually deliver $m$.
- No more blocking...
Changes to ACP-SB:

- Use UTRB instead of SB to broadcast decisions
- When a participant times out waiting for a decision message, just abort instead of using a termination protocol
- The second point above means no more blocking in ACP
UTRB1 – Simple UTRB

// broadcaster executes:
send [DLV: m] to all processes in G
deliver m

// process p !≠ broadcaster in G executes upon (first receipt of [DLV: m])
    send [DLV: m] to all processes in G
    deliver m
ACP-UTRB1 In Action

Coordinator receives votes as before...
P₂ broadcasts COMMIT before it goes down, or it could not have delivered the COMMIT message.
Performance

- **Modular:** \( \text{cost} = \text{cost of ACP} + \text{cost of instance of UTRB} \)
- **Time delay:** \( 2\delta + (F+1)\delta = (F+3)\delta \)
- **Message complexity:** \( 2n + n^2 \)
- \( n \) = number of participants
- \( F \) = maximum number of participants that may crash during this execution
Message-Efficient UTRB

- Use rotating coordinators
- Instead of each process broadcasting to all others, one process takes over in case of failure
- Adds delay for determining that the coordinator is down and for a process to notify the new coordinator
- Message complexity drops from $n^2 + n$ to $n + (f + 1)2n$
Other modifications to UTRB

- More time efficient – be pessimistic
  - Do not wait to be sure that the latest coordinator is down
  - Ask for the next coordinator after a much shorter wait
- Terminate early – detect when coordinator is down early and abort without having to wait the full timeout
Three-Phase Commit Protocol (3PC)

- Coordinator requests a vote
- If any process votes no, coordinator broadcasts **abort**
- If all processes vote yes, coordinator broadcasts **precommit**
- When all processes acknowledge the precommit, coordinator broadcasts **commit**
Coordinator requests a vote
3PC In Action

Participants respond with YES or NO
If all participants respond YES, coordinator broadcasts PRECOMMIT.
3PC In Action

Coordinator waits for acknowledgement

Coordinator

Awk

Awk

P₁

Awk

P₂
Now coordinator can broadcast COMMIT message
A crashed participant cannot recover and try to commit with other participants still waiting for a decision.

Failure of coordinator leaves participants to figure out action from one another.

Extra state of precommit means that can always occur, so no blocking.
Conclusion

- 2PC allows for atomic commitment of transactions, but is blocking
- Changing properties of the broadcast primitive creates a non-blocking protocol (APC-UTRB)
- Adding a phase can also prevent blocking (3PC)
- Is this really necessary? rarely