Distributed Transactions

Preliminaries

- Last topic: transactions in a single machine
- This topic: transactions across machines
- Distribution typically addresses two needs:
  - Split the work across multiple nodes
  - Provide more reliability by replication
  - Focus of 2PC and 3PC is the first reason: splitting the work across multiple nodes

Model

- For each distributed transaction T:
  - one coordinator
  - a set of participants

- Coordinator knows participants; participants don't necessarily know each other

- Each process has access to a Distributed Transaction Log (DT Log) on stable storage

The setup

- Each process has an input value, vote: Yes, No
  - Input could be based on program logic
  - Or it could be based on a local optimistic concurrency control (OCC) check

- Each process has to compute an output value decision: Commit, Abort

A digression: OCC

- Many variants of OCC, but here is a canonical version
- Transactions are assigned a txn number at completion
- When a transaction enters the system, note the highest committed txn number (start)
- When it is ready to commit, note the txn number (fin)
- OCC check for transaction t:
  - check that the write sets of all transactions from start to fin don’t intersect with the read set of the t
  - check that the write sets of ongoing transactions don’t intersect with either the read or write sets of t
  - if checks succeed, assign the next txn number to t & commit

Atomic Commit Specification

AC-1: All processes that reach a decision reach the <u>same</u> one.

AC-2: A process cannot <u>reverse</u> its decision after it has reached one.

AC-3: The Commit decision can only be reached if <u>all</u> processes vote Yes.

AC-4: If there are no failures and all processes vote Yes, then the decision <u>will</u> be Commit.

AC-5: If all failures are repaired and there are no more failures, then all processes <u>will</u> eventually decide.

Failures

- What are the different classes/types of failures in a distributed system?
- What guarantees should we aim to provide in building fault-tolerant distributed systems?

2-Phase Commit

Coordinator c

Participant p_i

I. sends VOTE-REQ to all participants

2-Phase Commit

Coordinator c

I. sends VOTE-REQ to all participants

Participant p_i

II. sends $vote_i$ to Coordinator

if $vote_i$ = NO then

$decide_i$:= ABORT

halt

2-Phase Commit

Coordinator c

I. sends VOTE-REQ to all participants

III. if (all votes YES) then
    $decide_c$:= COMMIT
    send COMMIT to all
else
    $decide_c$:= ABORT
    send ABORT to all who voted YES
halt

Participant p_i

II. sends $vote_i$ to Coordinator
    if $vote_i$ = NO then
        $decide_i$:= ABORT
        halt

2-Phase Commit

Coordinator c

I. sends VOTE-REQ to all participants

III. if (all votes YES) then
    $decide_c$:= COMMIT
    send COMMIT to all
else
    $decide_c$:= ABORT
    send ABORT to all who voted YES
halt

Participant p_i

II. sends $vote_i$ to Coordinator
    if $vote_i$ = NO then
        $decide_i$:= ABORT
        halt

IV. if received COMMIT then
        $decide_i$:= COMMIT
    else
        $decide_i$:= ABORT
    halt

- How do we deal with different types of failures?

Timeout actions

Processes are waiting on steps 2, 3, and 4

Step 2 p_i is waiting for VOTE-REQ from coordinator

Step 3 Coordinator is waiting for vote from participants

Step 4 p_i (who voted YES) is waiting for COMMIT or ABORT

Termination protocols

I. Wait for coordinator to recover
   - It always works, since the coordinator is never uncertain
   - may block recovering process unnecessarily

II. Ask other participants

Logging actions

1. When coord sends VOTE-REQ, it writes START-2PC to its DT Log
2. When p_i is ready to vote YES,
   - writes YES to DT Log
   - sends YES to coord
3. When p_i is ready to vote NO, it writes ABORT to DT Log
4. When c is ready to decide COMMIT, it writes COMMIT to DT Log before sending COMMIT to participants
5. When it is ready to decide ABORT, it writes ABORT to DT Log
6. After p_i receives decision value, it writes it to DT Log

p recovers

1. When coordinator sends VOTE-REQ, it writes START-2PC to its DT Log
2. When participant is ready to vote Yes, writes Yes to DT Log before sending yes to coordinator (writes also list of participants)
   When participant is ready to vote No, it writes ABORT to DT Log
3. When coordinator is ready to decide COMMIT, it writes COMMIT to DT Log before sending COMMIT to participants
   When coordinator is ready to decide ABORT, it writes ABORT to DT Log
4. After participant receives decision value, it writes it to DT Log

- if DT Log contains START-2PC, then $p = c$:
  - if DT Log contains a decision value, then decide accordingly
  - else decide ABORT
- otherwise, p is a participant:
  - if DT Log contains a decision value, then decide accordingly
  - else if it does not contain a Yes vote, decide ABORT
  - else (Yes but no decision) run a termination protocol

- What are the strengths/weaknesses of 2PC?

Blocking and uncertainty

Why does uncertainty lead to blocking?

- An uncertain process does not know whether it can safely decide COMMIT or ABORT because some of the processes it cannot reach could have decided either

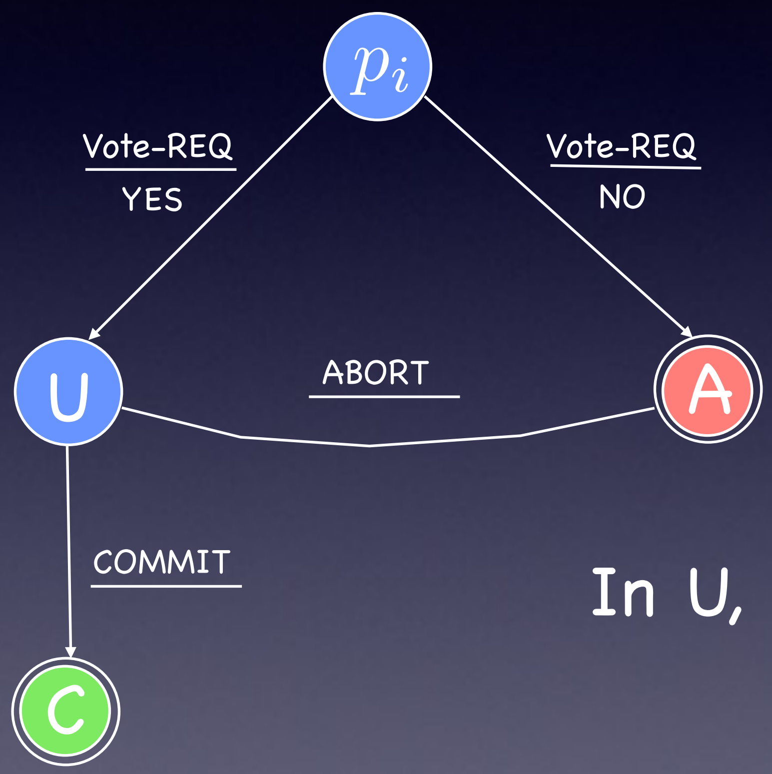
Non-blocking Property (desired!)

If any operational process is uncertain, then no process has decided COMMIT

Key Insight for 3-PC

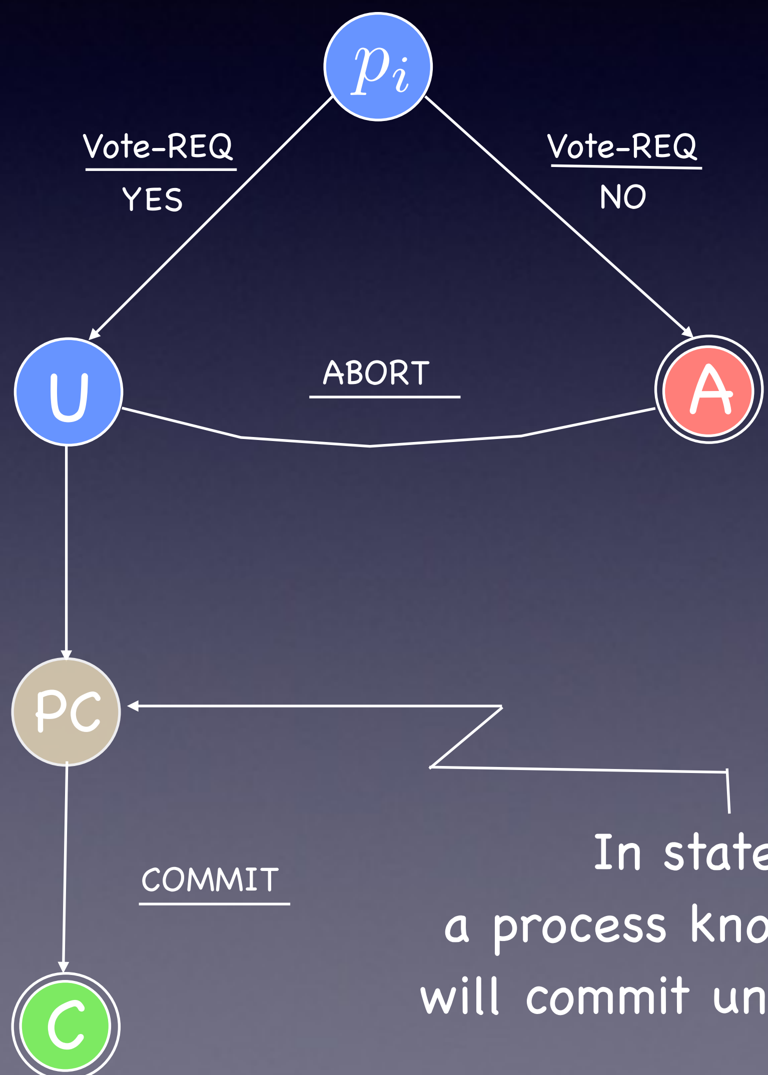
- Cannot abort unless we know that no one has committed
- We need an algorithm that lets us infer the state of failed nodes
  - Introduce an additional state that helps us in our reasoning
  - But start with the assumption that there are no communication failures

2PC Revisited



In U, both A and C are reachable!

2PC Revisited



In state PC
a process knows that it
will commit unless it fails

Coordinator Failure

- Elect new coordinator and have it collect the state of the system
- If any node is committed, then send commit messages to all other nodes
- If all nodes are uncertain, what should we do?

3PC: The Protocol

Dale Skeen (1982)

I. c sends VOTE-REQ to all participants.

II. When p_i receives a VOTE-REQ, it responds by sending a vote to c
if $vote_i$ = No, then $decide_i$:= ABORT and p_i halts.

III. c collects votes from all.
if all votes are Yes, then c sends PRECOMMIT to all
else $decide_c$:= ABORT; sends ABORT to all who voted Yes halts

IV. if p_i receives PRECOMMIT then it sends ACK to c

V. c collects ACKs from all.
When all ACKs have been received, $decide_c$:= COMMIT;
c sends COMMIT to all.

VI. When p_i receives COMMIT, p_i sets $decide_i$:= COMMIT and halts.

Termination protocol: Process states

At any time while running 3 PC, each participant can be in exactly one of these 4 states:

Aborted	Not voted, voted NO, received ABORT
Uncertain	Voted YES, not received PRECOMMIT
Committable	Received PRECOMMIT, not COMMIT
Committed	Received COMMIT

Not all states are compatible

	Aborted	Uncertain	Committable	Committed
Aborted	Y	Y	N	N
Uncertain	Y	Y	Y	N
Committable	N	Y	Y	Y
Committed	N	N	Y	Y

Failures

- Things to worry about:
  - timeouts: participant failure/coordinator failure
  - recovering participant
  - total failures

Timeout Actions

Processes are waiting on steps 2, 3, 4, 5, and 6

Step 2 p_i is waiting for VOTE-REQ from coordinator	Step 3 Coordinator is waiting for vote from participants
Step 4 p_i waits for PRECOMMIT	Step 5 Coordinator waits for ACKs
Step 6 p_i waits for COMMIT	

Timeout Actions

Processes are waiting on steps 2, 3, 4, 5, and 6

Step 2 p_i is waiting for VOTE-REQ from coordinator Exactly as in 2PC	Step 3 Coordinator is waiting for vote from participants Exactly as in 2PC
Step 4 p_i waits for PRECOMMIT Run some Termination protocol	Step 5 Coordinator waits for ACKs Coordinator sends COMMIT
Step 6 p_i waits for COMMIT Run some Termination protocol	

Termination protocol

- When p_i times out, it starts an election protocol to elect a new coordinator
- The new coordinator sends STATE-REQ to all processes that participated in the election
- The new coordinator collects the states and follows a termination rule

TR1. if some process decided ABORT, then?

TR2. if some process decided COMMIT, then?

TR3. if all processes that reported state are uncertain, then?

TR4. if some process is committable, but none committed, then?

Termination protocol

- When p_i times out, it starts an election protocol to elect a new coordinator
- The new coordinator sends STATE-REQ to all processes that participated in the election
- The new coordinator collects the states and follows a termination rule

TR1. if some process decided ABORT, then
  decide ABORT
  send ABORT to all
  halt

TR2. if some process decided COMMIT, then
  decide COMMIT
  send COMMIT to all
  halt

TR3. if all processes that reported state are uncertain, then
  decide ABORT
  send ABORT to all
  halt

TR4. if some process is committable, but none committed, then
  send PRECOMMIT to uncertain processes
  wait for ACKs
  send COMMIT to all
  halt

Discussion

- What are the strengths/weaknesses of 3PC?