

Two Applications of Paxos

Consider two other protocols for asynchronous systems:

1. Atomic commit.
 - 2 phase commit is blocking in the face of one (or two) well-placed failures.
 - 3 phase commit requires leader election (which is a perfect failure detector).
2. Primary backup.
 - Useful for nondeterministic state machines.

Paxos: As presented

Phase 1:

- a) A proposer selects a proposal number n and sends a *prepare* request with number n to a majority of acceptors.
- b) If the acceptor receives a *prepare* request with number n greater than any that of any *prepare* request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.

Phase 2:

- a) If the proposer receives a response to its *prepare* requests (numbered n) from a majority of acceptors, then it sends an *accept* request to each of those acceptors for a proposal numbered n with a value v , where v is the value of the highest-numbered proposal among the responses, or is any value if the responses report no proposals.
- b) If an acceptor receives an *accept* request for a proposal numbered n , it accepts the proposal unless it has already responded to a *prepare* request having a number greater than n .

Paxos: A Small Twist

Let there be a *distinguished proposer*, which we'll call the *leader*. It chooses proposal number 0. Any other proposer will choose a proposal number that is larger than 0, and do so only because it suspects the leader to be faulty.

The leader does not execute Phase 1. Instead, it starts out by sending an *accept* request with its preferred value. Call this the *ballot 0 message* for this Paxos protocol

Any new leader, of course, starts with Phase 1.

We'll have the leader take on role as the *distinguished learner* as well.

Paxos Commit

- There are *transaction managers (TM)* and *resource managers (RM)*.
 - In two phase commit, there is only one TM that collects the abort/commit votes from the RMs, decides the outcome, and disseminates the result to the RMs.
 - In the approach we discussed earlier in making 2PC nonblocking, we let the first phase unchanged and had the TM use the standard execute the second phase with a reliable broadcast protocol.
 - If we instead have *both phases* done via consensus, we can reduce the message delay.

Paxos Commit: Basics

- We will have $2t+1$ acceptors, which are assuming the role of the TM in 2PC.
- There is an instance of Paxos $Pax(r)$ for each RM r , which is used to reach agreement on the decision r makes to commit or abort.
 - If the decision of each consensus is *commit* then the transaction commits; otherwise, the transaction aborts.
 - For each consensus, the RM are the learners (recall that the leader is the distinguished learner).

Paxos Commit: I

- The same set of acceptors and the same leader is used for all instances of Paxos. All RM know the identity of the acceptors, and the RM and acceptors know the identity of the leader.
- The ballot 0 message is defined to be sent by the leader, but in fact any process could send it. Since RM r knows what the desired outcome should be, have r send the ballot 0 message for $Pax(r)$.

Paxos Commit: II

- Execution of Paxos Commit normally starts when some RM decides to commit and sends a *BeginCommit* message to the leader.
 - The leader sends a *Prepare* message to all of the other RMs.
 - Each RM r sends a ballot 0 message in $Pax(r)$.
 - For each instance of Paxos, each acceptor sends a message to the leader (in its role as the distinguished learner) when they accept the ballot 0 message.
 - The leader knows the outcome of $Pax(r)$ when it receives $t+1$ of these messages. As the distinguished learner, it forwards this outcome to all of the RM.

Paxos Commit: III

- Each acceptor can be message efficient and bundle all of the n messages to the leader into one physical message.
- The leader can also distill all of the messages it forwards to the RM into a single *Commit* or *Abort* message.

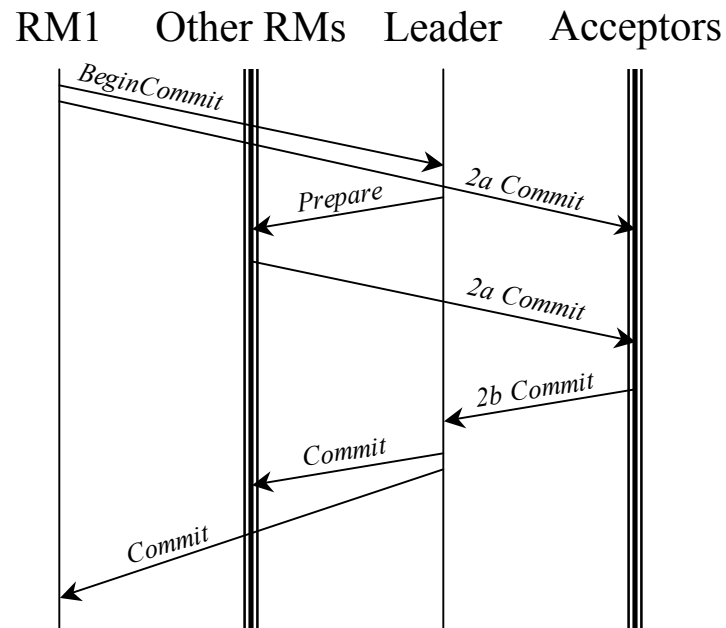
Paxos Commit: IV

- The leader may tire of waiting for the decision from some $Pax(r)$.
 - It can then start Phase 1 with a larger ballot number. If it finds the outcome not constrained, then it sends the value *abort* in Phase 2.
- $Pax(r)$ can decide *commit* only if r sent a ballot 0 message with *commit*.
 - So, Paxos can short circuit and inform all processes that the transaction has aborted if r sends a ballot 0 message with *abort*.
 - The same short circuit cannot be used if the leader sends *abort* in Phase 2a, since the result of $Pax(r)$ may still be *commit*. (how?)

Paxos Commit: V

- A new leader starts if the current leader is suspected of having crashed.
 - Can be done with a separate election protocol.
 - Can be done by having an RM r that tires of waiting for the result of $Pax(r)$.

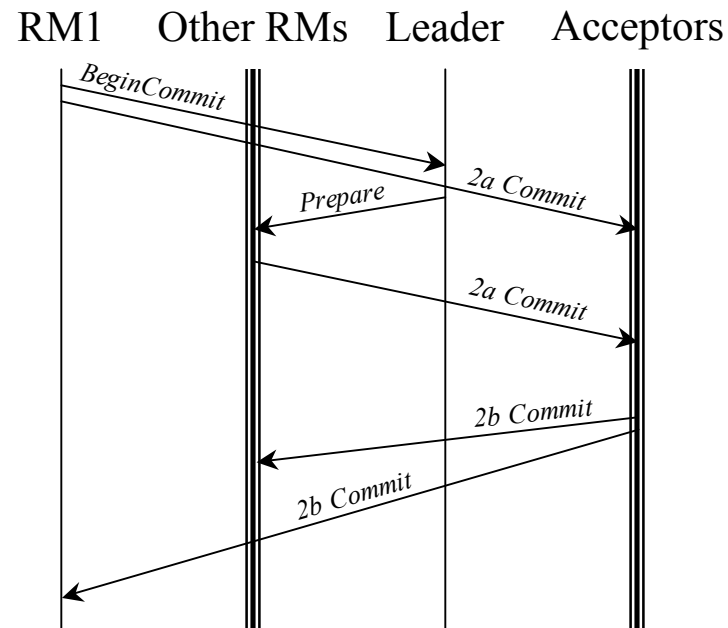
Cost: I



5 message delays, $(n+1)(t+3) - 4$ messages (with the leader an acceptor).

If each acceptor is on the same node as an RM, and the leader on RM1, then $n(t+3) - 3$ messages, but still 5 message delays.

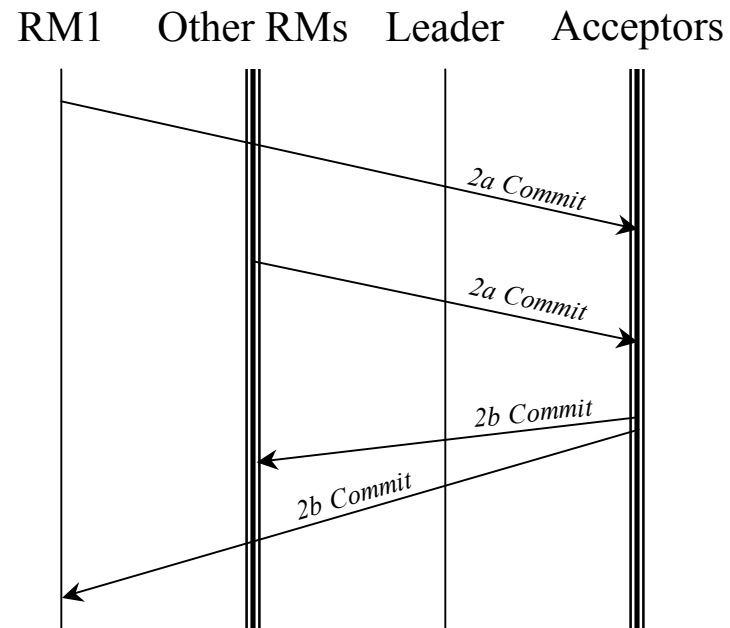
Cost: II



4 message delays, $n(2t+3) - 1$ messages

If each acceptor is on the same node as an RM, and the leader on RM1, then $(n-1)(2t+3)$ messages, but still 4 message delays.

Cost: III



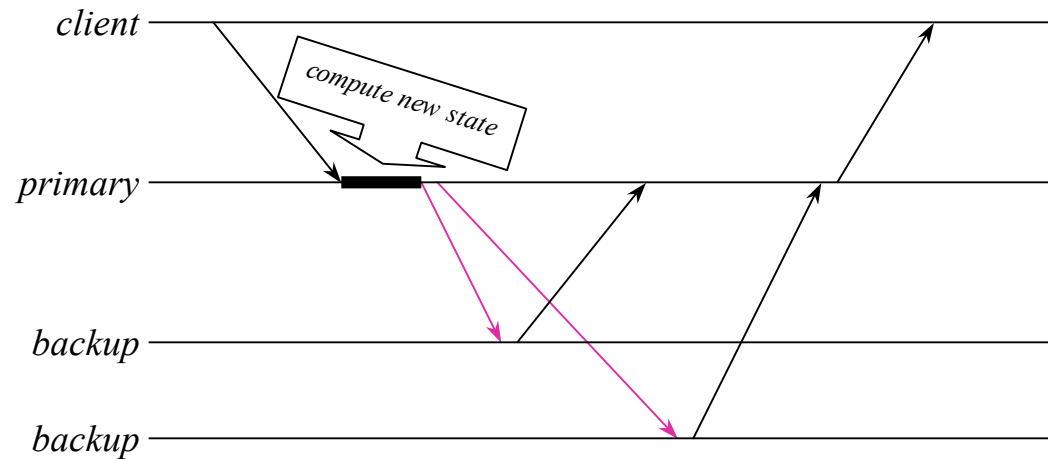
2 message delays, which is optimal

writes to stable storage?

Nondeterminism

- The state machine approach requires *deterministic* state machines.
 - Explicit nondeterminism.
 - Implicit nondeterminism.
- Primary-backup protocols can be used for nondeterministic servers.
 - Benign failures.
 - State update.

Primary-Backup in Synchronous Environment



Applying to Paxos

- Use the same structure as state machines using Paxos, but have the leader behave as in primary-backup: it computes the new state.
 - Store the state as a *transaction*: it can be discarded, leaving the server in the previous state.
 - Upon reaching consensus, each server applies the required state update.
 - If a server that computed a new state did not win the consensus, then it first aborts the transaction; else commits.

Cost
