Storage Systems

NPTEL Course

Jan 2012

(Lecture 33)

K. Gopinath

Indian Institute of Science

Summary of 2PC and 3PC

- 2PC: *ready(T) from all*: log *commit(T)*; send same to all C_k (logged)
  - Actually: flush the transaction and commit the state, and then msg
    - It cannot be msg followed by commit (coord could crash in betw)
  - But having committed, it cannot be unrolled
  - If just after commit, coord crashed, nobody knows what it did
  - Hence, block (FLP result applies)
- 3PC: *ready(T)* from every site: *precommit(T)* to log & to each site (log)
  - Do not flush the transaction and commit the state to avoid 2PC block
  - Instead inform others what you plan to do
  - But now we get stuck in the “2 generals problem”
    - Map coord to 1 general and others to 2nd general
  - Keep optimistically trying to solve the “2 generals problem” till success
    - By leader election to try new coord
  - What if leader election a stumbling block?

2-phase commit

Transaction T initiated at site S_i and txn coord there C_i.
When subT completed at all sites (all sites inform C_i), start 2PC protocol

- **Phase 1**: C_i logs *prepare(T)*; sends *prepare(T)* to all C_k
  C_k: on receiving prepare msg, either
  does not commit: logs *no(T)* and sends *abort(T)* to C_i
  commits: logs *ready(T)* with all changes to log onto stable storage and sends *ready(T)* to C_i
- **Phase 2**: C_i receives response to prepare msg from all C_k or timeout:
  ready(T) from all: log *commit(T)*; send same to all C_k (logged)
  else: log *abort(T)*; send same to all C_k (logged)
  each C_k sends *ack(T)*

C_i receives acks from all: logs *complete(T)*
ready(T) from a site: will follow coord's order to commit or abort

3-phase commit

To avoid blocking in limited cases:

- eg. no netw partition; atmost K sites can fail & atleast K+1 sites up

Provide preliminary info about fate of T thru a precommit phase

Assume failures detected by coord or sites reliably

- **Phase1**: same as 2PC
- **Phase2**: C_i receives responses to prepare msg from all C_k or timeout:
  any site *abort(T)* or no response from a site until timeout: *abort(T)* to all
  ready(T) from every site: *precommit(T)* to log & to each site to its log
  ack sent to coord from each site whether abort or precommit (logged)
- **Phase3**: only executed if precommit in Phase2
  coord waits till atleast K acks: logs *commit(T)* & sends it to each to its log

ready(T): site's promise to follow coord's decision

precommit(T): coord's promise to commit

Relation Betw AC and Consensus

- Differences betw AC and diff versions of consensus concern
- the decisions reached by faulty processes; and
- the strength of the conditions required

AC attainable only under the assumption that process failures are benign

Can prove

- AC 2,3,4 *imply* WV but not the converse
- With “no-catastrophe” axiom (NC): all failures repaired and no new failures for a sufficient period of time, then AC1, AC4 and NC *imply* A

AC conditions stronger than WV, assuming NC

Paxos (Lamport)

- Clients, proposers, acceptors, and learners
  - client requests distr system (wlock in dfs), waits for resp
  - proposer attempts to ratify a proposed decision value
  - by collecting acceptances from a majority of acceptors
  - ratification observed by learners
- Only 1 proposal can get votes of maj of acceptors
- Have to avoid deadlock when there are more than two proposals or when some processes fail
  - Proposer can effectively restart the protocol by issuing a new proposal (thus dealing with lockups)
  - can release acceptors from their old votes if provable that old votes were for a value that cannot get a majority any time soon

Alg

- Before taking a vote, a proposer checks by sending a prepare(*n*) message to all acceptors (*n*: proposal #)
- An acceptor responds with a promise never to accept any proposal with a # less than *n*
  - older proposals don't suddenly get ratified
- *with* highest-# proposal that acceptor has accepted
  - so that proposer can substitute this value for its own, in case previous value was in fact ratified
- If proposer receives a response from a majority of the acceptors, the proposer then does a second phase of voting where it sends an accept(*n*, *v*) to all acceptors and wins if it receives a majority of votes.

Paxos Safety Properties

- Holds regardless of the pattern of failures
- Non-triviality
  - Only proposed values can be learned
- Consistency
  - At most one value can be learned (i.e., two different learners cannot learn different values)
- Liveness(C; L)
  - If value C has been proposed, then *eventually* learner L will learn some value (if sufficient processors remain non-faulty)

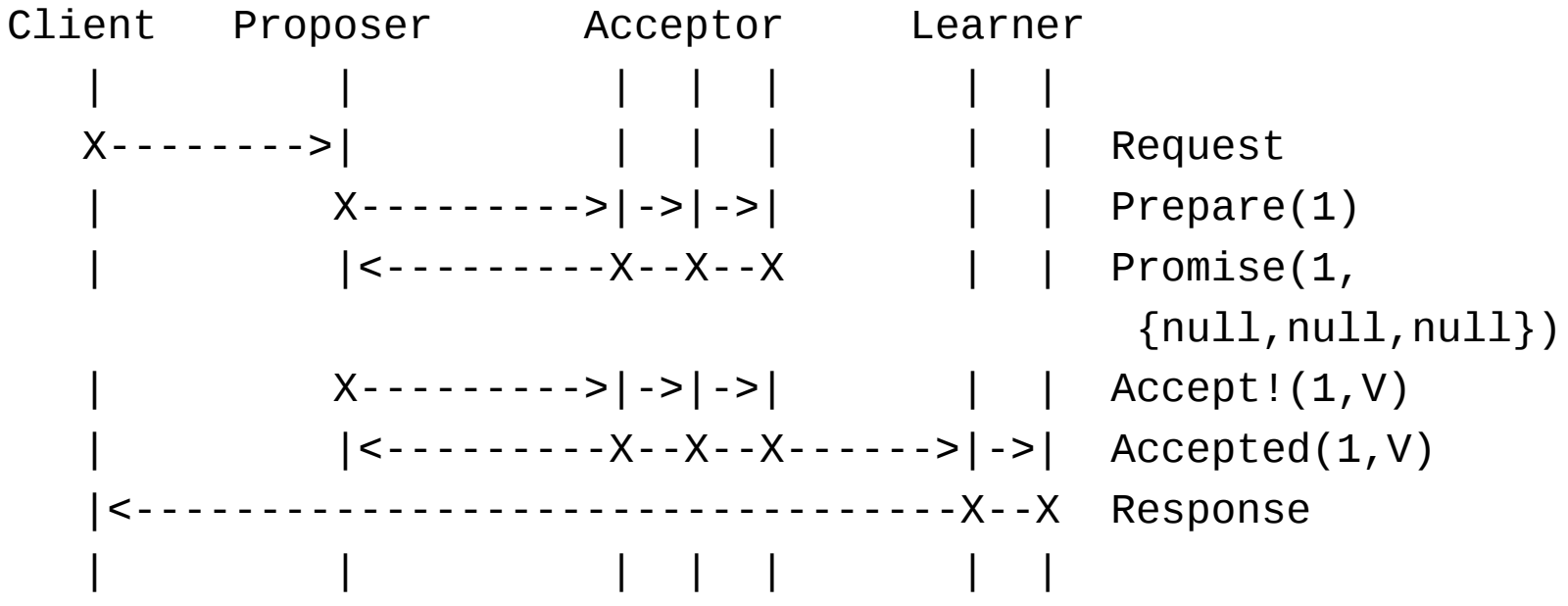
Basic Paxos

- Each instance decides a single output value
- A Proposer P should not initiate Paxos if it cannot communicate with at least a Quorum of Acceptors Q
- A successful round has two phases:
  - Phase 1a: Prepare
    - P (“leader”) sends a prepare message proposal N (> any previous one by P) to Q
  - Phase 1b: Promise
    - If N > any prev proposal # recd from any Proposer by an Acceptor, then Acceptor must return a promise to ignore all future proposals < N.
    - If an Acceptor accepted a proposal previously, it must include prev proposal number and its value in its response
    - Otherwise, an Acceptor can ignore the received proposal.

- Phase 2a: Accept Request
  - If P receives enough promises from Q, it needs to set a value to its proposal.
  - If any Acceptors in Q had previously accepted any proposals, P must set value to that of the highest proposal number reported by Q. If none in Q had accepted any, then P may choose any value
  - Sends an Accept Request(N) message to Q with value
- Phase 2b: Accepted
  - If an Acceptor receives an Accept Request(N), it must accept it iff it has not already promised to only consider proposals > N.
  - If accept, send corresp value v thru an Accepted message to P and every Learner. Else, can ignore Accept Request(N).

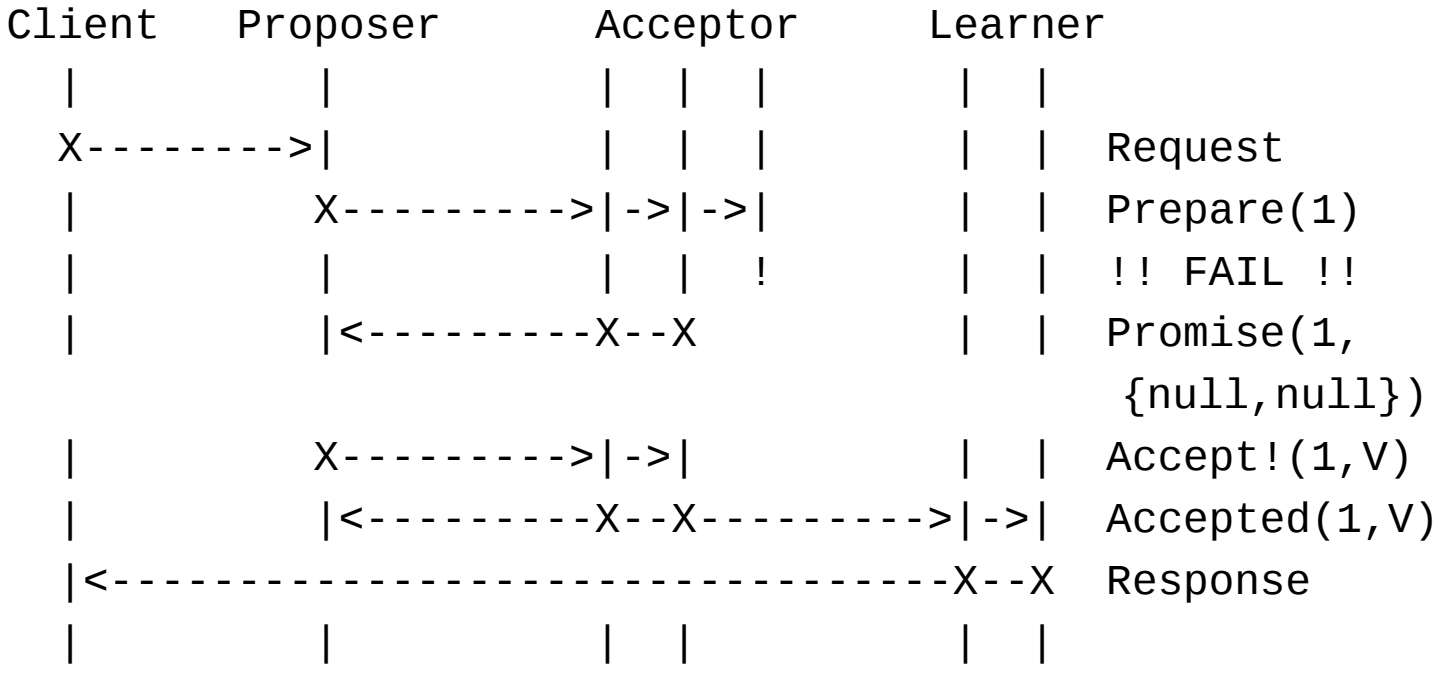
Rounds fail when multiple Proposers send conflicting Prepare messages, or when P does not receive a Quorum of responses (Promise or Accepted); another round must be started with a higher proposal number.

First Round itself Successful

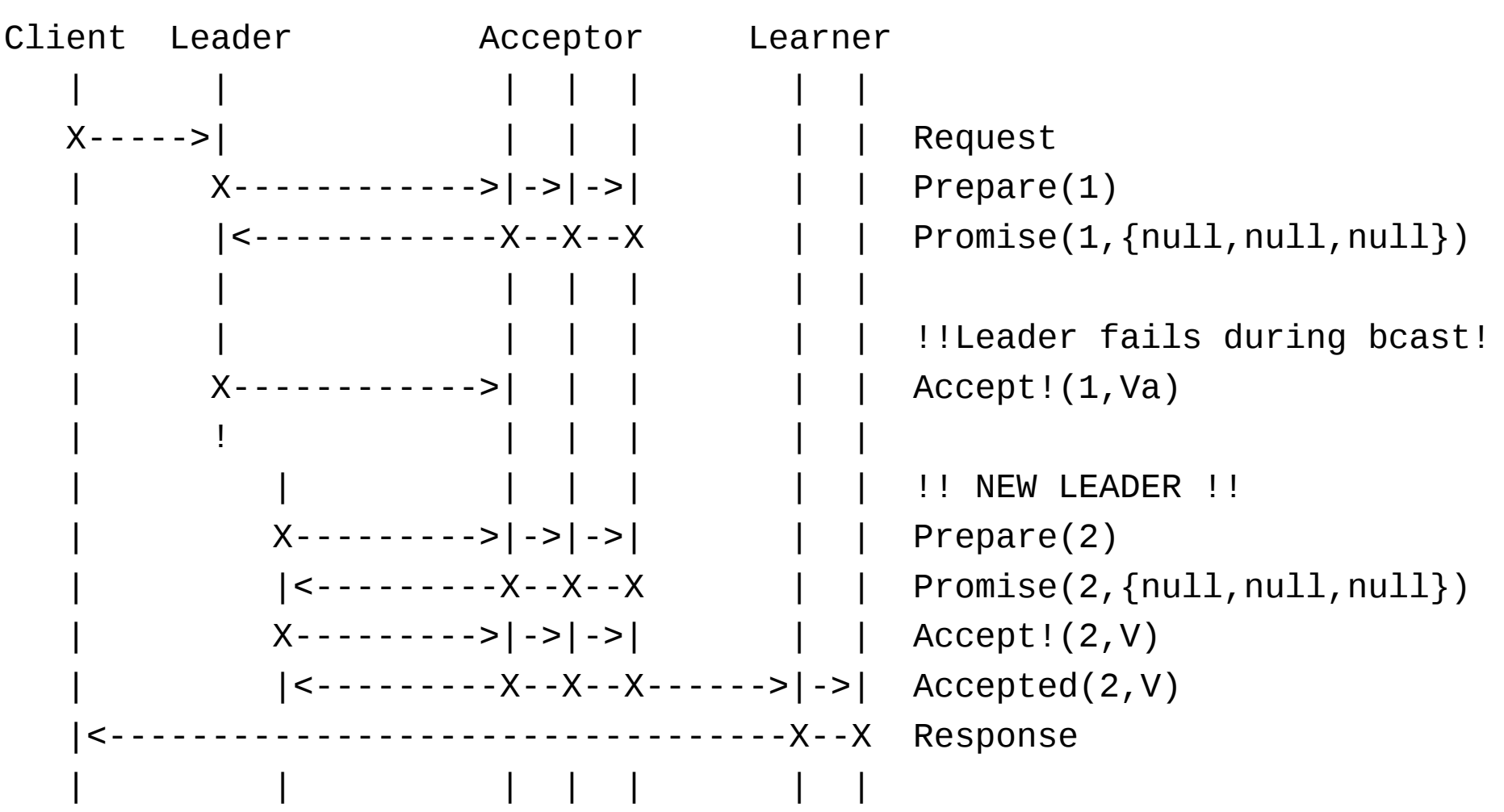


(diags from Wikipedia)

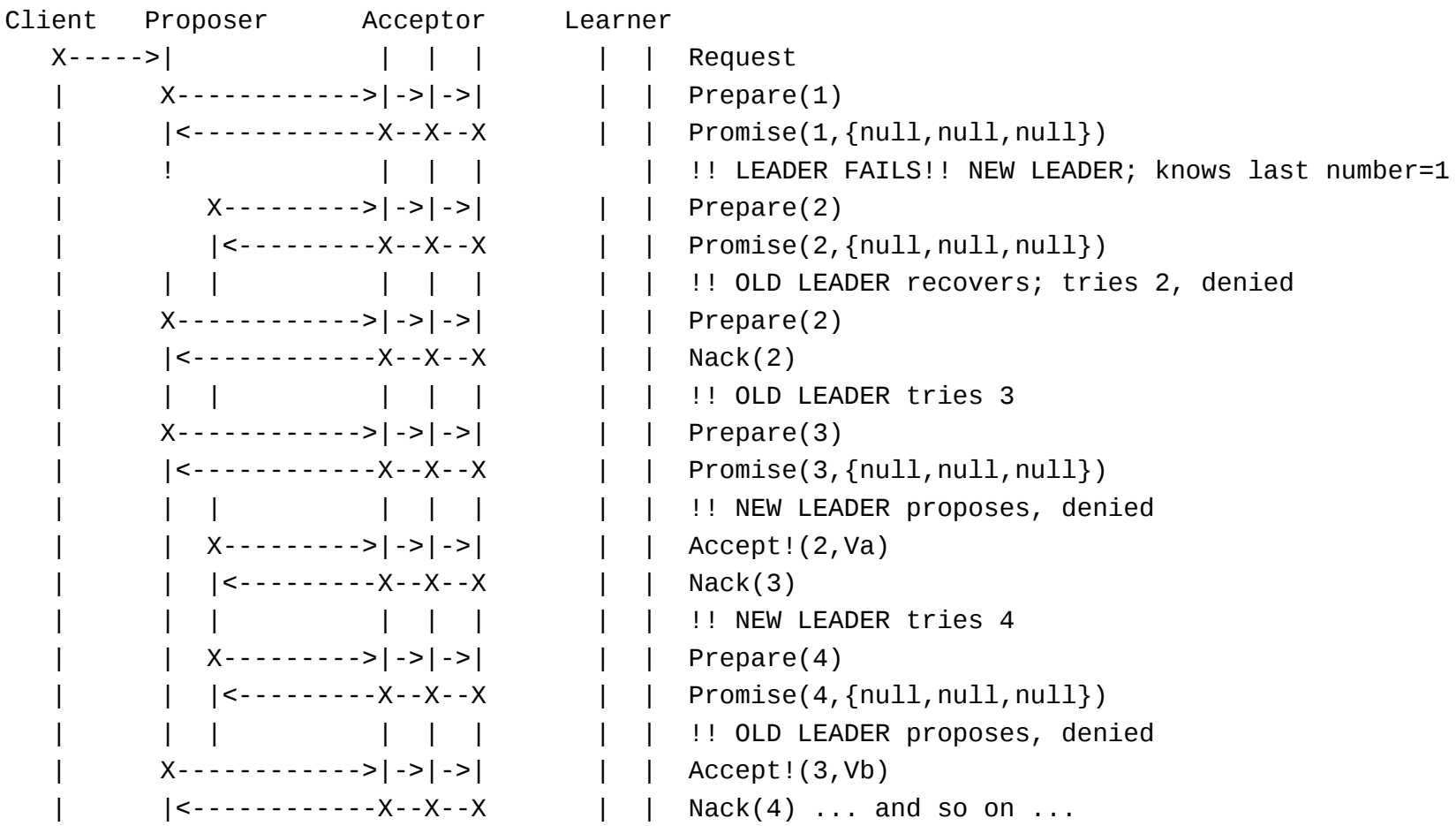
Failure of Acceptor (Quorum size = 2 Acceptors)



Failure of Proposer (re-election not shown, 1 instance, 2 rounds)



Dueling Proposers (one instance, four unsuccessful rounds)



Summarizing Paxos

- Run by a set of proposers (non deterministic & without persistent storage) that guide a set of acceptors (deterministic with persistent storage that survives crashes) to achieve consensus
- Correct no matter how many simultaneous proposers there are and no matter how often proposers or acceptors fail and recover, how slow they are, or how many messages are lost, delayed, or duplicated
- Terminates if there is a single proposer for a long enough time during which the leader can talk to a majority of acceptors twice
- May not terminate if there are always too many proposers (guaranteed termination is impossible)
- Set of proposers/acceptors fixed for a single run of alg

- For liveness, can combine Paxos with a sloppy timeout-based algorithm for choosing a single leader
- If sloppy algorithm leaves us with no leader or more than one leader for a time, the consensus algorithm may not terminate during that time
- But if the sloppy algorithm *ever* produces a single leader for long enough the algorithm will terminate, no matter how messy things were earlier
- To get a really efficient system, usually necessary to use leases as well

Sloppy algorithm for choosing a single leader:

- proposers have clocks, max time to send, receive, & process a msg known
- every potential leader that is up broadcasts its name.
- A proposer becomes the leader one round-trip time after doing a broadcast unless it has received a broadcast of a bigger name