Paxos Made Simple

Presented by Nakul Bhasin
Rahul Choudhari
Dharmesh Jagadish

Paxos

- Algorithm given by Leslie Lamport in 2001.
- Paxos is a family of protocols for solving consensus in a network of unreliable processors (a fault tolerant distributed system).

Consensus Algorithm

- A consensus algorithm ensures that a single one among the proposed values is chosen.
- If no value is proposed, then no value should be chosen.
- If a value has been chosen, then all the other processes should be able to learn the chosen value.

Safety Requirements for Consensus

- Only a value that has been proposed may be chosen.
- Only a single value is chosen
- A process never learns that a value has been chosen unless it actually has been.

Goal of Consensus algorithm

- Ensure that some proposed value is eventually chosen
- If a value has been chosen, then a process can eventually learn the value.

Roles in the consensus algorithm

- Three roles in the consensus algorithm performed by three classes of agents
  - Proposers- propose a value to be chosen
  - Acceptors- decide which value to choose
  - Learners- learn which value was chosen
- A single process may act as more than one agent.
- Agents can communicate by sending messages

Assumptions in Model

- asynchronous & non-Byzantine model
  - Agents operate at arbitrary speed.
  - Agents may fail after a value is chosen and restarted
  - Messages:
    - can take long to be delivered
      - can be duplicated
      - can be lost
      - but not corrupted.

Approaches for Choosing a Value

Single Acceptor(Easiest way): choose a value to have a single acceptor agent.

- A proposer sends a proposal to the acceptor.
- The acceptor who chooses the first proposed value that it receives.

.

Choosing a value (contd.)

Using multiple acceptor agents:

- Proposer sends a proposed value to a set of acceptors.
- An acceptor may accept the proposed value.
- The value is chosen when a large enough set of acceptors have accepted it.

Requirement P1

- Paxos should work even if only one proposal is ever made (ignoring message loss).

- P1:

An acceptor must accept the first proposal that it receives.

But what if multiple proposals are made simultaneously?

- There might not be a majority of acceptors that accepted the same proposal.
- Therefore acceptors must be able to accept more than one proposal.

- P1 isn't enough.

Solution for P1

- keeping track of the different proposal:
  < proposal number ; value >
- Different proposals have different numbers.

- Value chosen when single proposal with that value accepted by a majority of the acceptors

- Guarantees safety property that only a single value is chosen

- Allow multiple proposals to be chosen

Requirement P2

We don't really need to require that only one proposal is chosen, as long as:

P2:If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.

- condition P2 guarantees that only a single value is chosen

Requirement $P2^a$

- To be chosen, a proposal must be accepted by at least one acceptor.

  So, we can satisfy P2 by satisfying:

$P2^a$:

If a proposal with value v is chosen, then every higher numbered proposal accepted by any acceptor has value v.

- If $P2^a$ holds, then P2 holds

Requirement $P2^b$

- Maintaining both P1 and $P2^a$ requires strengthening $P2^a$ to:

$P2^b$:

If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.

- If $P2^b$ holds, then $P2^a$ must hold ,i.e.,

$P2^b \rightarrow P2^a \rightarrow P2$

$P2^c$:

- For any v and n, if a proposal with value v and number n is issued then there is a set S consisting of a majority of acceptors such that

  either

(a) no acceptor in S has accepted any proposal numbered less than n or

(b) v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S

Proposers Algorithm

STEP 1: Prepare request:

Send by the proposer with proposal number n to each member of some set of acceptors, asking it to respond with:

(a) A promise never again to accept a proposal numbered less than n, and

(b) The proposal with the highest number less than n that it has accepted.

Proposers Algorithm contd:

Step 2 :

- proposer receives the responses from a majority of the acceptors,
- issue a proposal with number n and value v

v->value of the highest-numbered proposal among the responses

- This is an accept request send by proposer

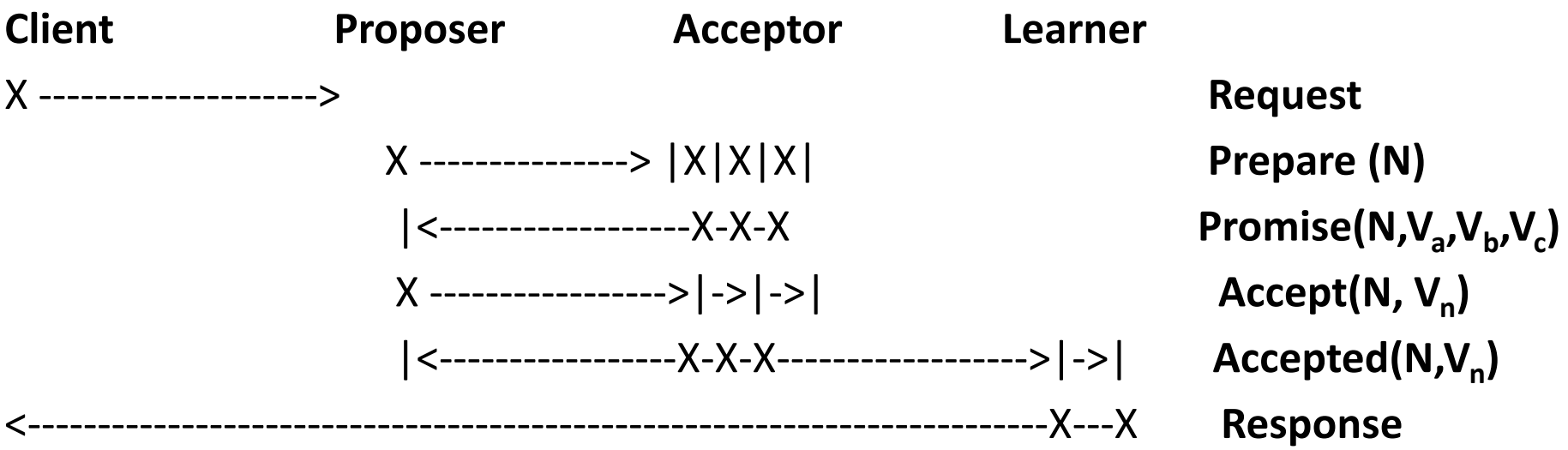
What Acceptors do

- receive two kinds of requests from proposers: prepare requests and accept requests
- It can always respond to a prepare request
- It can respond to an accept request, if the following requirement is satisfied:

$P1^a$:An acceptor can accept a proposal numbered n iff it has not responded to a prepare request having a number greater than n.

- Together, $P1^a$ and $P2^c$ give us the consensus algorithm.
- $P2^c$ tells proposers what proposals they can issue.
- $P1^a$ tells acceptors what proposals they can accept.
- Only one value can ever be chosen.
- Only a majority of acceptors need to accept a proposal to choose that value.

Message Flow in Paxos



Paxos Algorithm phase 1

Algorithm has two phases:

Phase 1 [*prepare request*]

- *Proposer chooses number n; sends a request to some majority of* acceptors asking each for:

  (a) a promise not to accept proposals < *n, and*

  (b) the highest-numbered proposal < *n that it has* already accepted, if any

Paxos Algorithm: phase 2

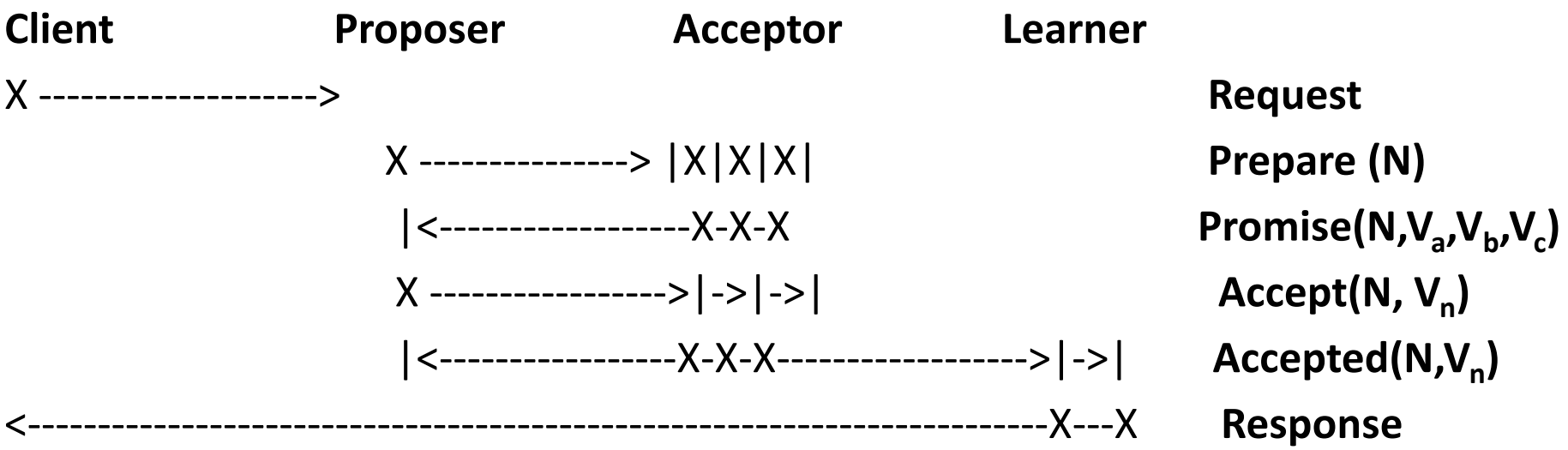
Phase 2 [accept phase]

If majority of acceptors respond, issue proposal *n* whose value is:

(a) the value of the highest-numbered proposal among the responses, or

(b) anything we want if there weren't any such proposals.

Message Flow in Paxos



Example

- Proposers are p_1 and p_2.
- Acceptors are a_1, a_2, and a_3.
- p_1 sends prepare for proposal 1 to a_1 and a_2.
- a_1 and a_2 reply to p_1.
- p_2 sends prepare for proposal 2 to a_2 and a_3.
- a_2 and a_3 reply to p_2.

Example (contd.)

- p_1 sends accept request to a_1 and a_2 for proposal 1 with value “X”
  - p_1 got to select which value to propose.
- a_1 accepts proposal 1.
- a_2 does not accept proposal 1.
  - a_2 promised p_2 it wouldn't accept proposals < 2

Example contd.

- p_2 sends accept request to a_2 and a_3 for proposal 2 with value “Y”
  - p_2 also got to select which value to propose.
- a_2 accepts proposal 2.
- a_3 accepts proposal 2.
- $\{a_2, a_3\}$ is a majority of acceptors, so proposal 2 is chosen.
  - The chosen value is “Y”

Example contd.

- p_1 *sends prepare* for proposal 3 to a_1 and a_2.
- a_1 *replies;* it last accepted proposal 1 for "X".
- a_2 *replies;* it last accepted proposal 2 for "Y".
- p_1 sends *accept request* to a_1 and a_2 for proposal 3 with value "Y"
  – Value must match the one from proposal 2.
- a_1 and a_2 accept proposal 3.

Learning a chosen value

How Do Learners Learn?

- Each acceptor informs each learner.

$$n_{responses} = n_{learners} \times n_{acceptors}$$

- Each acceptor informs a distinguished learner, who relays to the other learners
- acceptors respond with their acceptances to a distinguished Learner

$$n_{responses} = n_{learners} + n_{acceptors}$$

Learning contd.

- assumption of non-Byzantine failures
- requires lots of messages to be sent
- extra round required for all the learners to discover the chosen value
- single point of failure
- less reliable, since the distinguished learner could fail

Learner algorithm (contd.)

- hybrid solution: set of distinguished learners
- Compromise with a set of distinguished Learners
  - Limits number of messages needed.
  - All distinguished learners need to fail to cause a problem.

Optimization

- acceptor ignore to respond to a prepare request numbered n when a prepare m, with m > n has been already responded.
- It should ignore a prepare request for a proposal it has already accepted
- An acceptor should inform a proposal when it has delivered a proposal, while the acceptor already responded to an higher one.

Progress

- Solution: allow only a *distinguished proposer* to prepare and issue proposals.
  - Proposers send their proposals to the distinguished proposer, who organizes them.
  - Doubles as distinguished learner.
- Can we avoid single-point-of-failure?
  - New distinguished proposer is elected if the current one fails.
  - Two or more processes can think they're distinguished without compromising correctness (but can prevent progress).

Implementation

- Each process plays the role of proposer, acceptor, and learner
- Leader election for the distinguished proposer and the distinguished learner
- An acceptor records its intended response in stable storage before actually sending the response.

Implementation contd.

- Guarantee that no two proposals are ever issued with the same number

  -> every proposer choose their numbers from disjoint sets of numbers

  -> each proposer remembers (in stable storage) the highest-numbered proposal it has tried to issue, and begins phase 1 with a higher proposal number than any it has already used.

So what is it good for?

Is a consensus on a single arbitrary value actually *useful?*

Yes!

The Paxos consensus algorithm can be used to construct a distributed system of state machines.

State Machine

Distributed system: collection of clients issuing commands to central server.

- If single central server fails, system fails.
- **State machine approach:** implement fault-tolerant service by replicating servers and coordinating client interactions with server replicas.

Implementing a State Machine

1. Place State Machine copies on multiple, independent servers.
2. Receive client requests (Inputs) to State Machine.
3. Choose ordering for Inputs.
4. Execute Inputs in chosen order on each server.
5. Respond to clients with Output from the State Machine.
6. Monitor replicas for differences in State or Output.

Implementing a State Machine

- Paxos can be used to implement distributed state machine:
  - values being agreed upon are commands to execute (commands by client).
  - one instance of Paxos executed for each command (value chosen by i^{th} instance is the i^{th} state machine command).
  - infinite number of instances executed simultaneously.
  - leader is elected to be the distinguished proposer and distinguished learner.

Isn't Infinity Too Big?

- Phase 1 doesn‘t require knowing the value being proposed
- Leader can execute Phase 1 immediately, wait until something to propose for Phase 2.
- Phase 1 for all instances can be done with a single prepare request.
- E.g “this is a prepare request for proposal n for all instances > 30.”

Implementing State Machine contd.

- If leader fails, new leader appointed.
- New leader already a learner in previous instances(first i instances)
- Sends prepare request for instances > i, uses same proposal no.
- If values had been chosen for any instances > i, the replies will include those proposals.

State Machine Example

- New leader knows commands 1-134, 138, and 139.
- Sends *prepare requests for instances 135*-137 and 140+.
- Receives replies with existing proposals for instances 135 and 140.
- Issues accept requests for instances 135 and 140 with the appropriate values.

Filling in the Gaps

- There may be “gaps” --

  instances where no value had been proposed.

  - State machines can't execute command i until all commands < i executed
- Leader chooses values to propose to fill those gaps in.
- Safest choice: no-op command.

State Machine Example (2)

- Commands 136, 137, and 141+ are still undecided.
- To fill the gap, leader issues *accept requests* for instances 136 and 137 with a value of "no-op".
- When the next command request arrives, the leader issues an *accept request for instance* 141 with that command as the value.
- Then issue 142, 143, ...

Application of Paxos

- Google Megastore used for 3 billion writes and 20 billion read transactions daily.
- It uses Paxos to manage synchronous replication between datacenters.
- Provides the highest level of availability for reads and writes at the cost of higher-latency writes.
- Paxos is also used to perform write operations.

Conclusion

- Safety is a guarantee in Paxos despite asynchrony.
- Once a the distinguish leader is elected, liveness is guaranteed.
- According to paper: phase 2 has the minimum possible cost of any algorithm for reaching agreement in presence of faults.