Distributed Systems
Master of Science in Engineering in Computer Science

AA 2018/2019

LECTURE 9: CONSENSUS - PAXOS

Impossibility Result

Impossibility of Consensus in asynchronous systems in the presence also of a single crash

Fisher, Lynch e Patterson (FLP result).
Ref: Journal of the ACM, Vol. 32, No. 2, April 1985.

Paxos

The Paxos family of algorithms was introduced in 1999 to provide a viable solution to consensus in asynchronous settings:

- Safety is <u>always</u> guaranteed
- The algorithm makes some progress (liveness) only when the network behaves in a "good way" for long enough periods of time (partial synchrony).

Paxos – recap about properties

Let us think about what the protocol should do.

- 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

Paxos – System model and basic assumptions

- Agents operate at arbitrary speed, may fail by stopping (and may restart).
    - Observation: Since all agents may fail after a value is chosen and then restart, a solution is impossible unless some information can be remembered by an agent that has failed and restarted.
- Messages can take arbitrarily long to be delivered, can be duplicated, and can be lost, but they are not corrupted.

Paxos - Actors

Actors in the basic Paxos protocol

- Proposer: proposes a value.
- Acceptors: processes that must commit on a final decided value.
- Learners: passively assist to the decision and obtain the final decided value.

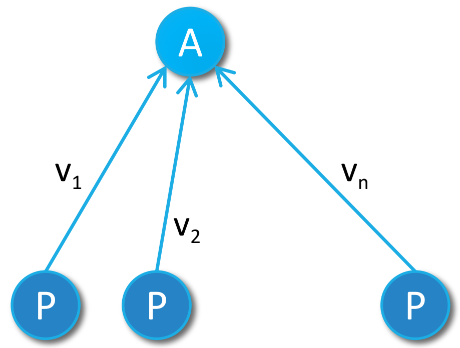
Paxos: choosing a value

The easiest way to solve the problem is to have a single acceptor

- A proposer sends a proposal to the acceptor
- The acceptor choses the first proposed value it receives

What if the only acceptor fails ?

=> We must have more than one acceptor



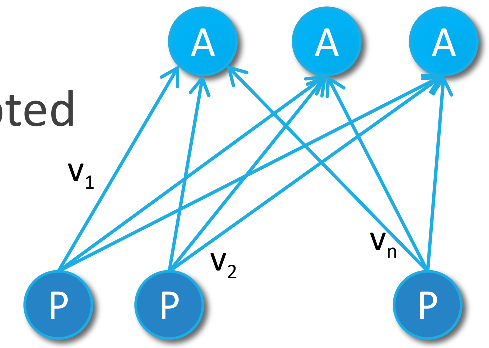
Paxos: choosing a value

Multiple acceptors

- A proposer sends a proposal to a set of acceptors
- An acceptor MAY accept the proposed value
- A value is chosen when a majority of acceptors accept it

The majority is needed to guarantee that only a value is accepted

An acceptor may accept at most one value



Paxos: choosing a value

If you assume no failures and no message loss

- We want a value to be chosen even if only one value is proposed by a single proposer

P1: An acceptor must accept the first proposal it receives

Problem: if several values are concurrently proposed by different proposers, none of them could reach the needed majority. We have a sort of deadlock.

This implies that an acceptor should accept multiple proposals

Paxos: choosing a value

Solution

- We keep track of different proposal assigning them a proposal number that is unique (total order of proposals)
- A value is chosen when a single proposal with that value has been accepted by a majority of the acceptors (chosen proposal).
- We can allow multiple proposal to be accepted, but all accepted proposals must have the same value.

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

Paxos: choosing a value

For a value **v** to be chosen, a proposal containing it must have been accepted by at least one acceptor, thus:

P2a: if a proposal with value v is accepted, every higher-numbered proposal that is accepted by any acceptor has value v

One more problem: What if a proposal with value **v** is accepted while an acceptor **c** never saw it ?

A new proposer could wake up and propose to **c** a different value **v'** that **c** must accept due to P1

Paxos: choosing a value

We must slightly modify P2a to take into account this case:

P2b: if a proposal with value v is chosen, every higher-numbered proposal issued by any proposer has value v

How can we guarantee P2b ?

Paxos: choosing a value

Assume a proposal **m** with value **v** has been accepted

We should guarantee that any proposal **n**, with **n**>**m** has value **v**

We could prove it by induction on **n** assuming that every proposal with number in [**m**, **n-1**] has value **v**

For **m** to be accepted there is a set C of acceptors (a majority) that accept it

Paxos: choosing a value

Therefore, the assumption that **m** has been accepted implies that:

- Every acceptor in C has accepted a proposal with number in [**m**,**n-1**] and every proposal in this range accepted by an acceptor has value **v**

Paxos: choosing a value

Given the intersection among majorities, we can conclude that a proposal numbered n has value by ensuring that the following invariant is maintained:

P2c: 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.

By maintaining P2c we satisfy P2b.

Paxos: how to ensure to choose a value

P2c can be maintained by asking to a proposer that wants to propose a value numbered **n** to learn the highest-numbered value with number less than **n** that

- has been accepted
- or will by accepted

by any acceptor in a majority.

Paxos: learning about proposed values

Learning accepted values is easy

Instead of trying to predict the future, ask acceptors to promise they will not accept proposals numbered less than **n**

Paxos: learning about proposed values

The protocol has two main phases:

- Phase 1: prepare request ↔ response
- Phase 2: accept request ↔ response

Paxos

Phase 1

- 1) A proposer chooses a new proposal version number n , and sends a prepare request (PREPARE,n) to a majority of acceptors:
  - (a) Can I make a proposal with number n ?
  - (b) if yes, do you suggest some value for my proposal?

Paxos

Phase 1

- 2) If an acceptor receives a prepare request (PREPARE, n) and it will answer sending:
  - (a) responds with a promise not to accept any more proposals numbered less than n.
  - (b) suggests the value v' of the highest-number proposal that it has accepted if any, else ⊥

  - In particular, it will reply with the highest number less than n that is has accepted
    - (ACK, n, n', v') if it exists
    - or (ACK, n, ⊥ , ⊥) if not

Paxos

Phase 1

- 2) If an acceptor receives a prepare request (PREPARE, n) with n lower than n’ from any prepare request it has already responded, sends out (NACK, n’)
    - (a) responds with a denial to proceed with the agreement as the proposal is too old.

Paxos

Phase 2

- 3) If the proposer receives responses from a majority of the acceptors, then it can issue an accept request (ACCEPT, n , v) with number n and value v:
  - (a) n is the number that appears in the prepare request.
  - (b) v is the value of the highest-numbered proposal among the responses (or the proposer’s own proposal if none was received).

Paxos

Phase 2

- 4) If the acceptor receives an accept request (ACCEPT, n , v) , it accepts the proposal unless it has already responded to a prepare request having a number greater than n.

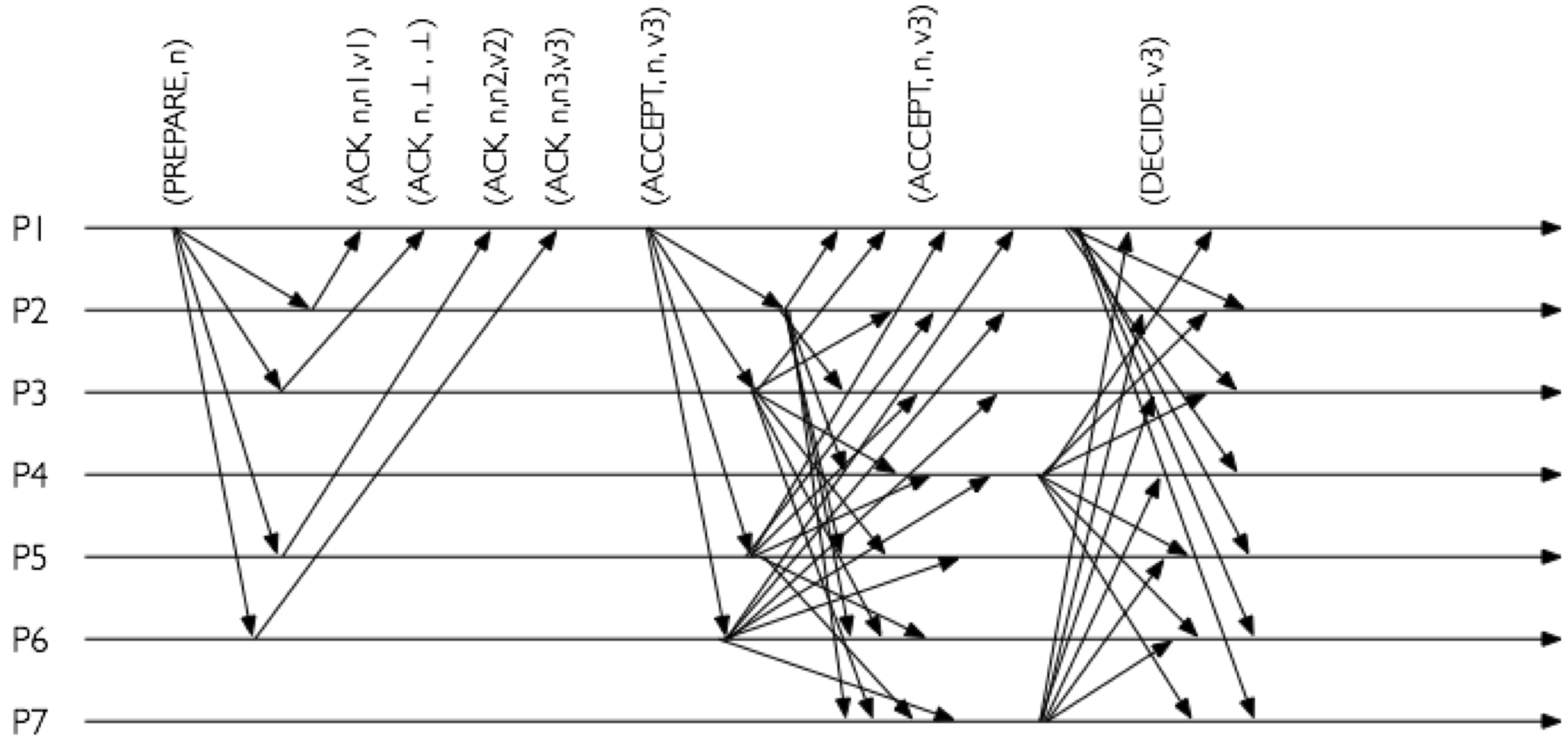
Paxos

Learning the decided value

- Whenever acceptor accepts a proposal, respond to all learners (ACCEPT, n, v).
- Learner receives (ACCEPT, n, v) from a majority of acceptors, decides v, and sends (DECIDE, v) to all other learners.
- Learners receive (DECIDE, v), decide v

Paxos

Run example:



Paxos

Liveness is not guaranteed (due to FLP)

- Imagine a scenario with two competing proposers

Omissions delay the protocol but do not block it

Proposers are “elected” using a leader election protocol

- Protocol becomes a 2-phase commit with a 3- phase commit when leader fails

Paxos

Is Paxos just a theoretical exercise ?

- Used but not widely. For example, Google uses Paxos in their lock server. Yahoo for ZooKeeper
- One issue is that Paxos gets complex if we need to reconfigure it to change the set of nodes running the protocol
- Another problem is that other more scalable alternatives are available

References

[1] L. Lamport “*Paxos Made Simple*”, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/paxos-simple-Copy.pdf

And for very brave students… ☺

[2] L. Lamport. The part-time parliament. ACM Transactions on Computer Systems, 16(2):133–169, May 1998.