University of Freiburg, Germany
Department of Computer Science

Distributed Systems

Chapter 5 Paxos

Christian Schindelhauer

05.06.2014

5.1: Introduction

- Paxos was proposed by Leslie Lamport to resolve consensus
  - in an asynchronous distributed systems
  - with time failures
  - without byzantine failures
- It is very influential and there is now a family of Paxos protocols

Literature

Funny written essay which introduces Paxos as fake history	Lamport, Leslie (1998) *The Part-Time Parliament* ACM Transactions on Computer Systems 16 (2): 133–169
Straight-forward write up of the same protocol by the same author in order to prove the simplicity of the algorithm	Lamport, Leslie (2001) *Paxos Made Simple* ACM SIGACT News (Distributed Computing Column) 32, 4

5.2: Consensus

- Processes need to agree on the same value
- It is not important which process wins the race

Safety Properties of Paxos

- **Nontriviality:** The resulting value must be proposed by a process
- **Consistency:** All learners agree only on one value
- **Liveness:** If a learner accepts a value, then eventually all learners accept this value

- Paxos ensures these properties in the face of any (non-Byzantine) failures

5.2: Comparing Consensi

- We already discussed consensus problems

Classic Consensus Problem

- *Termination:* Eventually each correct process p_i is *decided* by setting variable d_i
- *Agreement:* The decision value d_i of all correct processes is the same
- *Integrity:* If all correct process proposed the same value v, then $d_i = v$ for all correct p_i

Safety Properties of Paxos

- *Nontriviality:* The resulting value must be proposed by a process
- *Consistency:* All learners agree only on one value
- *Liveness:* If a learner accepts a value, then eventually all learners accept this value

- What is the difference?

5.2: Comparing Consensi

- What is the difference?
  - Termination!
  - Classic consensus claims that all deciders eventually agree on the same value
- Paxos allows that a proposed value is *not* learned
  - Such a proposed value can be proposed later on
  - Perhaps it is learned then
- In the original Paxos paper a continuous series of decrees is envisaged
  - This can lead to a race condition which is never resolved
- However termination cannot be guaranteed in crash-failure systems!
  - No algorithm can reach (classic) consensus even if only **one processor** is faulty [Fischer, Lynch, Paterson 1985]
- The weakening of the assumptions in Paxos is a clever solution to this dilemma.

5.3: The Settings

- Processes
  - have different speed
  - have independent failures
  - may rejoin after failure without loss or damage of their memory (**new**)
  - cooperate, i.e. do not lie or try to attack the protocol
    - for non-cooperating processes there is the Byzantine Paxos protocol
- Communication
  - unicast messages
  - asynchronous timing model
    - may take arbitrarily long
    - message loss cannot be distinguished from message delay until the message arrives
  - messages can be lost, reordered, or duplicated
  - *but* messages are **not** corrupted
    - corrupted messages can be solved by Byzantine Paxos

5.4: State Machine and Counting

- The consensi are numbered uniquely
  - The numbering depends on the implementation
  - Each Proposer must increase its number
  - Concurrent Proposers must never use the same number
  - The numbering does not have to be contiguous
- If a consensus fails, then this corresponds to a **nop** operation (no operation)
- Missing numbers are counted as **nop**
- The Paxos protocols simulates a server
  - which is resolving conflicting operations
  - and assigns numbers to each operation

5.5: Leader Election

- is considered as an easy operation by Paxos.
- It is assumed that the Proposers live long enough active to elect a Leader, e.g. the process with the smallest ID
- If more than one Proposer believes to be the Leader
  - then the Paxos protocol is still consistent, i.e. safety is preserved.
  - but it may be stalled
- If no server is acting as leader, then no new commands will be proposed.
- Election of a single leader is needed only to ensure progress.

5.6: Roles

- Client
  - issues a *request* and waits for *response*
  - e.g. „write“-request on a distributed file server
- Acceptor
  - Acceptors work in *quorums*, a group which is voting on requests.
  - They issue responses and act like the fault-tolerant memory
  - accept only once.
- Proposer
  - tries to convince the Acceptors that the *request* is o.k.
  - coordinates conflicts
- Learner
  - act as replicators.
  - If a client request has been granted (and agreed upon) by the Acceptors, the learners take action
  - e.g. execute the request, send responses to the client
- Leader
  - is a distinguished Proposer
  - if more than one Proposer believe that they are leaders, this conflict needs to be resolved

Quorums and Choice

- Quorum
  - is the majority of participating acceptors
  - e.g. if five Acceptors participate, then a quorum is reached, if three of the five agree.
  - for even number $2n$ of processors $n+1$ must agree to reach a quorum,
  - for odd number $2n-1$ of processors n must agree.
- Quorum can be generalized:
  - A Quorum is a set S of Acceptors
  - Each pair of Quorums must have an non-empty intersection
- Choice
  - If values are conflicting, then any value may be chosen
  - However, the value must have occurred in the most recent round
  - The value is chosen by the Leader by any function, e.g. majority or maximum
- In some implementations processes may play more than one role, e.g. Proposer, Acceptor and Learner
- This reduces the number of messages and does not harm the correctness

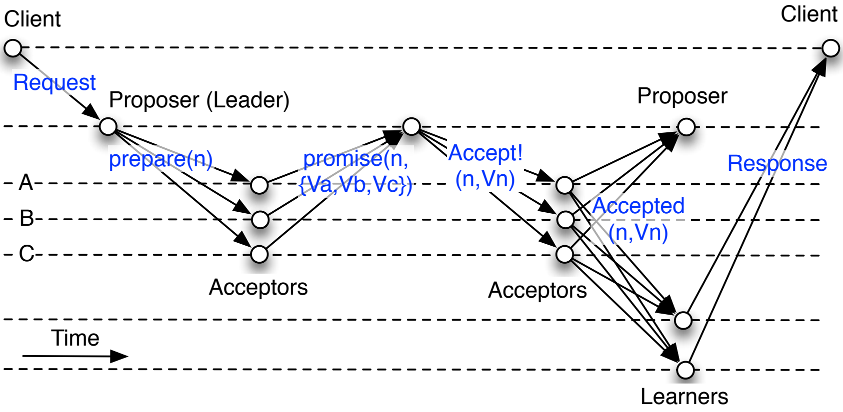
Basic Paxos - First Phase

- Phase 1a: Prepare
  - The Proposer (the Leader) selects a proposal number n and sends a **prepare** message to a Quorum of Acceptors
- Phase 1b: Promise
  - If the proposal number n is larger than any previous proposal
    - then each Acceptor promises not to accept proposals with a proposal number less than n
    - and sends a **promise** message including proposal number and value
  - otherwise the Acceptor sends a denial
  - Also each Acceptor sends the value and number of its last accepted or promised proposal to the Proposer

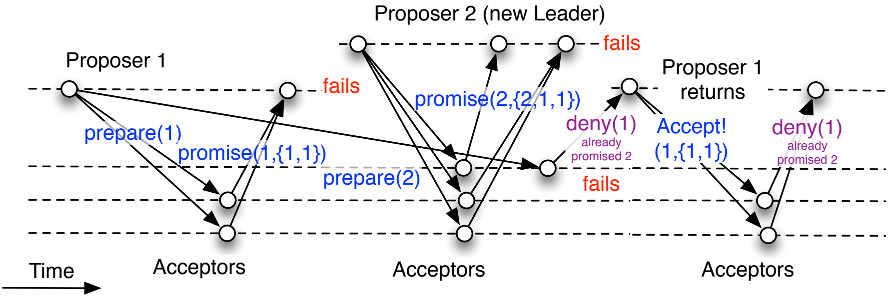
Basic Paxos - Second Phase

- Phase 2a: Accept!
  - If the Proposer receives (positive) responses from a Quorum of Acceptors
    - it may **choose** a value to be agreed upon
    - this value must be from the values of the Acceptors that have already accepted a value
    - otherwise the proposer can choose any value.
  - The Proposer sends an **accept!** message to a quorum of Acceptors including the chosen value
- Phase 2b: Accepted
  - If the Acceptor receives an **accept!** message for the most recent proposal it has promised,
    - it accepts the value
    - each Acceptor sends an **accepted** message to the proposer and every Learner.
  - otherwise it sends a denial and the last proposal number and value it has promised to accept

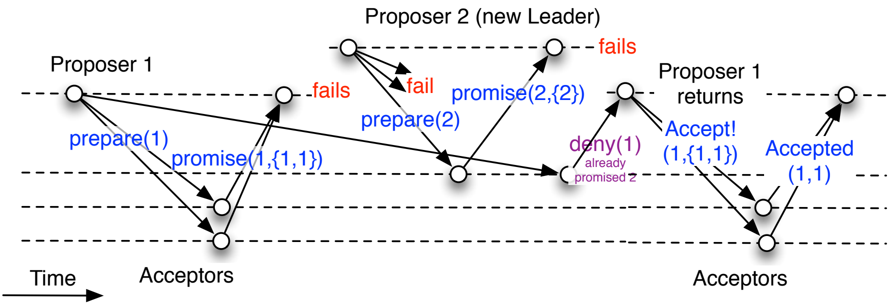
Basic Paxos — without Errors



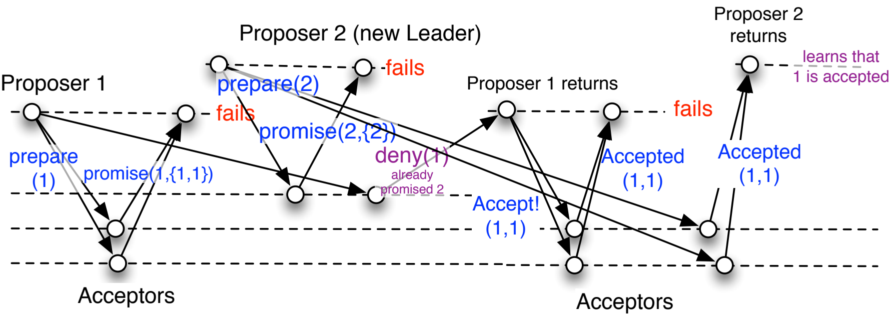
Basic Paxos — Failures and no Value Accepted



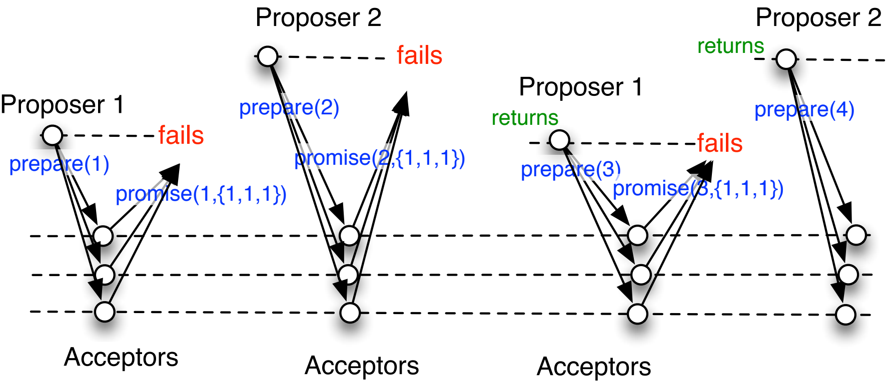
Basic Paxos — Failures and the First Value Accepted



Basic Paxos — Consistency in Time



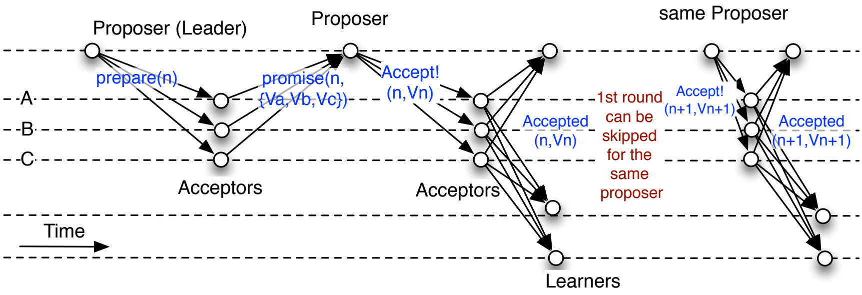
Basic Paxos — Termination not Guarranteed



Multi-Paxos

- Paxos can be optimized regarding Message Complexity
- The first round can be skipped if the proposer stays the same.
- Then, the previous 2nd round plays the role of the following 1st round.
- Only the proposer is allowed to skip the 2nd round who succeeded in the 1st round.
- This way, the delay reduces to two round and the number of messages reduce to the quorum
- This implementation is called *Multi-Paxos*

Multi-Paxos — Reducing the Delay and the Message Complexity



Further Optimizations

- Learners
  - A single distinguished Learner serves as relay and informs the other Learners when a value has been chosen
  - In most applications the role of the leader includes the role of the distinguished Learner
- Quorum communication
  - The leader may send *prepare* and *accept* only to a quorum
  - Other acceptors do not need to be bothered unless they are needed
- Hashing the value: Instead of sending the value, it suffices to send cryptographic secure hash values

Byzantine Paxos

- Byzantine Paxos deals with Byzantine Failures
- Here, the Client sends directly the proposal to the Acceptors
- The Acceptors exchange all received **prepare** or **accept!** messages and compute the Byzantine agreement
- The Learners wait for receiving $F + 1$ identical messages
- where F denotes the maximum number of Byzantine failures.
- The Learners respond to the client.

End of Section 5