DISTRIBUTED SYSTEMS
CSCI 4963/6963
10/30/2015
PAXOS

“This paper was first submitted in 1990, setting a personal record for publication delay that has since been broken by [60].”
The Significance of Paxos

• The Two Generals Problem and the FLP Impossibility result showed that it is there is no algorithm for consensus that “works” in “unreliable” systems.

• But many applications still need to solve consensus.

• Paxos provides a way to do this.
  • Doesn’t completely satisfy agreement, validity, and termination, but is good enough to be useful.

• Paxos has a reputation (undeservedly) for being very complicated and hard to understand.
Application of Paxos

• Paxos is used in Google’s distributed locking service, called Chubby.

• Chubby provides a mechanism for Google applications to serialize access to a resources.
  • GFS uses Chubby to appoint a master server
  • Big Table uses Chubby to elect a master server and to let the master keep track of its slaves.
  • GFS and Big Table also use Chubby for storage of important metadata.
The Story of Paxos

• “Early in [the last millennium], the Aegean island of Paxos was a thriving mercantile center”.

• Paxos had a parliamentary government, but its legislators were busy.
  • The part-time parliament.

• Parliament’s task was to determine the law of the land.
  • A sequence of decrees:
    • “The olive tax is 3 drachmas per ton.”
    • “Lamps must use only olive oil.”
    • …

• Each legislator maintained a ledger.
  • Ledger keeps ordered list of decrees.
  • Ledgers must be consistent.
    • No conflicting or missing entries.
The Story of Paxos (2)

• Legislators were willing to pass any decree that was proposed.

• Legislators would leave and enter the chamber whenever they felt like it, and they might not return.

• Poor acoustics in the chamber, so legislators communicate by sending messengers.
  • A messenger might forget he delivered a message and deliver it again.
  • A messenger might take a 6 month trip before delivering a message.
  • And a messenger may take a trip and not return, so the message is never delivered.
Progress Condition

• The Paxos devised a protocol that satisfied a progress condition:

• If a majority of legislators were in the chamber, and no one entered or left the chamber for sufficiently long, then

  (1) any decree proposed would eventually be passed

  (2) every decree that had been passed would eventually appear in the ledger of every legislator in the chamber
The Problem

- We need to store a log of events (decrees)

| Event 1 | Event 2 | Event 3 | Event 4 | Event 5 | .... |

- Want to replicate this log at multiple sites.
  - Need the event to appear in the same order in every log.
  - Need every event to eventually appear in every log.

- Challenges
  - Concurrency – different processes want to write to log at same time
  - Failures – processes and channels
System Model

- Processes may fail by crashing and may restart.
  - Both crash failures and crash/recovery processes.
  - Processes do not lose state when they crash and recover.
  - No Byzantine failures.

- Messaging is asynchronous.
  - Messages may take arbitrarily long to be delivered.
  - Messages can be duplicated.
  - Messages can be lost.
  - Messages cannot be corrupted.
A more straightforward description

• “Paxos Made Simple”
Leslie Lamport, ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001) 51-58.

• Abstract: “The Paxos algorithm, when presented in plain English, is very simple.”
The Paxos Algorithm

- Initial approach: consider the replicated log problem as a sequence of consensus problems, one per log position.

- The algorithm used to reach consensus on a single log entry is called the Synod Algorithm.

- Later: show how to make this initial approach more efficient – this is Paxos.
Requirements for Consensus Algorithm

• Safety
  • Only a value that has been proposed may be chosen (decided as consensus value).
  • Only a single value is chosen.
  • A process never learns that a value has been chosen unless it actually has been (no changing a decision).

• Liveness
  • No precise requirement
  • Goal is that eventually some proposed value is chosen, and then eventually a process can learn the chosen value.
Roles in the Consensus Algorithm

- **Proposers**: propose value to chosen (the consensus value)
- **Acceptors**: decide whether to accept a proposed value
- The value is *chosen* when enough acceptors accept it.
  - Enough = a majority
- **Learners**: must all learn the consensus value (after it is chosen)
  - For our discussion, Acceptors = Learners
Algorithm Properties

• (P1) An acceptor must accept the first value it receives.

• Why?
  • We want algorithm to work well when there are no competing proposals and no failures.

• Suppose two values are proposed (2m+1 total acceptors).
  • One is accepted by m acceptors.
  • The other is accepted by m+1 acceptors.
  • Single failure can make it impossible to learn which value was chosen.

• An acceptor must be able to accept more than one proposal.