8 Fault Tolerant Distributed Transactions : 2PC

x based on Weikum / Vossen; Valduriez / Öszu; Garcia-Molina ; Reuter/ Gray

8.3 Paxos consensus

Goals

Safety

- Consensus / data consistency in a distributed system similar to 2PC / 3PC
- Values may be proposed, but only a single value is chosen (unanimous decision)

Liveness

- Tolerate (non-malicious) failures.
- Some proposed value is eventually chosen.
- A chosen value can be learned (by others, perhaps recovering nodes)

Intro

- **Central part of Paxos** :

  Crash-consensus protocol that
  - is always safe.
  - may not terminate
    but terminates under some liberal constraints (basically: the system is alive with a sufficient number of nodes for a sufficient time)
- More General than 2PC /3PC,
  2PC may be seen as a special case.

Consensus

- N processes want to agree on a value
- Want to tolerate *F* faults
  Tolerate *F* processes stopping
  Tolerate *F* Messages delayed or lost
- If there are **less than *F* faults** in a window **then consensus** achieved.
- Note: no blocking
- Benign faults need *2F+1* "acceptors"
  stalls but safe if more than *F* faults

  Byzantine faults need *3F* "acceptors"

some slides based on Gray, Lamport

Paxos consensus in a nutshell

Roles

- **Processes** (proposers, TM,...)
- **Acceptors** (2F +1 if F faults to be tolerated)
- **Learner** (processes which have to get the consensus value, e.g. if they where down during consensus algorithm)

A thread (or OS process) may have **more than one role**, but mapping roles to threads is not a big deal.

Call the nodes with different roles **agent** (node)

Paxos leader

- Assume a **leader** process
- Election of a leader L or a substitute if L fails is a separate routine (non – trivial, leader election as such solves the agreement problem – impossible)
- Makes first phase of protocol simpler, but no principle difference to protocol without leader
- First leader given naturally by application in many cases - e.g. transaction mgr.

Paxos basics

n nodes, two phases:

(1) Prepare phase: leader sends (ballot number * "prepare") to all agents (nodes)

Wait for answer from agents a, which *encodes answer in earlier rounds* from a (!)

(2) **Accept phase**: leader sends a value to be accepted, *if quorum (more than n/2) has been reached*.
If proposed value is accepted by majority, leader announces the consensus value.

An agent accepts only, of it had not answered to a ballot with greater number.

Problem: many rounds can co-occur, depending on failures.

Paxos: properties

- Any proposal number must be unique. Obvious...
  how??
- Any two sets of agents have at least one agent in common.
- In accept phase the value proposed by a leader is the highest-numbered value proposed by agents in the prepare phase.
  Highest numbered: highest ballot (round) .

Paxus consensus

Group has a ***leader* known to all**
leader election is a subroutine

Process proposes
a value *v* to leader.

Leader sends proposal (phase 2)
(ballot, value) **to all acceptors**

Acceptors respond with:
max (ballot, value)
they have seen

If leader gets **no higher ballot**, and gets at least *F+1* responses then leader can announce (ballot, value)

Full protocol 3-phase

Phase 1:
Leader starts new ballot

Phase 2
Leader proposes value

Phase 3
If value accepted by *F+1* then value is accepted.
If not, leader tries to get majority value accepted.

Note: **Agents** are split into **proposers** and **acceptors.**

Paxos Commit

Obvious idea:

- Use more than one TM
- Have TM use Paxos consensus of *RMs prepared* with 2F+1 acceptors (in "consensus box")
- No blocking, if a TM fails the other one takes over and runs a consensus
- TMs and acceptors in the consensus box may (will) be typically the same.

Paxos Commit

More efficient idea:

2F+1 acceptors (~*2F+1* TMs)

Each RM leads a Paxos on: *I'm Prepared*.

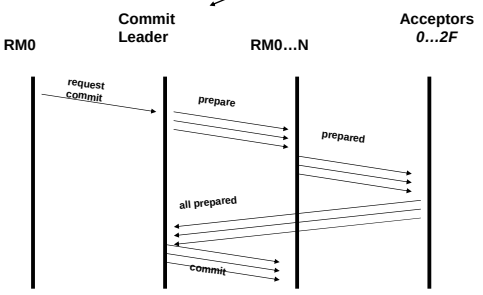
If *F+1* acceptors see all RMs prepared, then transaction committed.

2F(n+1) + 3n + 1 messages
5 message delays (one extra delay)
2 stable write delays.

F=0 3n +1 msg (2PC , not counting acks)

Paxos Commit

Stateless (!) TA-Mgr.



Why does it work? Many subtle cases...; proof exists

8.6 Paxos in practice

Paxos consensus not used for commit processing up to now.

- AACID (Availability + ACID) does not seem to be an issue in fixed networks
- Message overhead in mobile networks

Used for control of replication in
Google lock manager ("chubby")
(see next chapter on replication)
and a distributed file system.

Summary

Challenge of distributed transaction processing:
- Guarantee of isolation -> concurrency control
  -> basically local resource managers
- Atomicity in case of failure

Two phase commit: fault tolerant protocol
- Log records on stable storage essential
- Resource Managers may be blocked (coordinator fails)

Three phase commit: avoids blocking in case of site failure, not for communication failures

Paxos commit: tolerates f failures with 2f+1 acceptors, but may end up in live locks.

Many optimizations in practice

X/Open XA etc is the standard for Transaction Mgrs

Application Servers: relief application programmers from idiosyncratic program structure for distributed applications (and more)