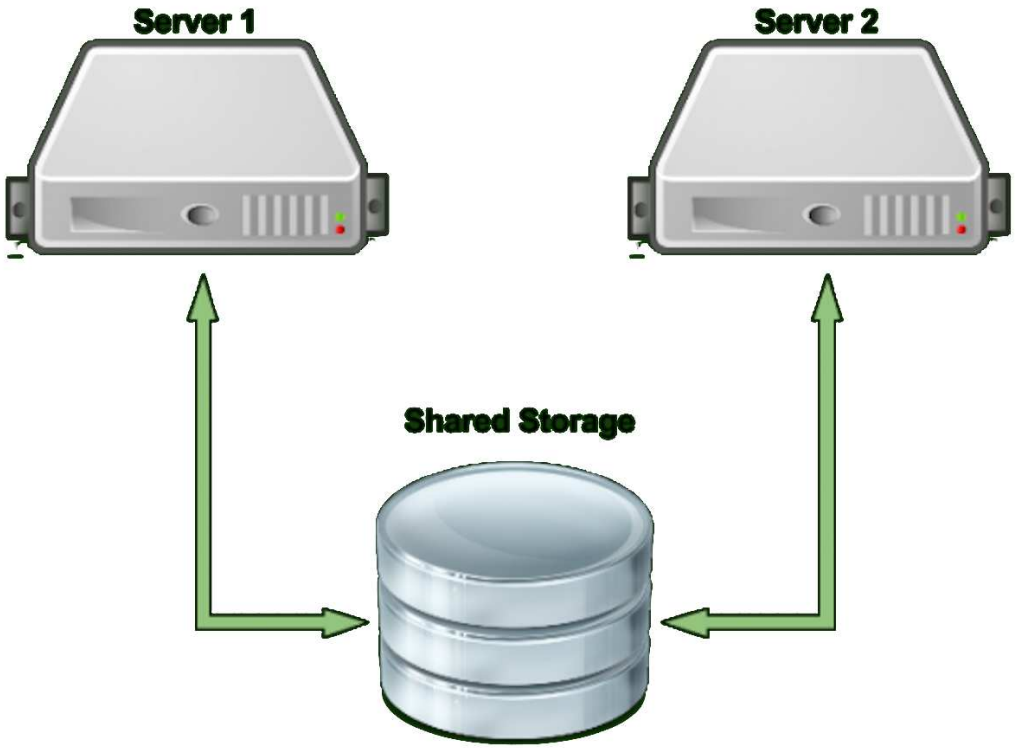
Disk Paxos and Δ-Leases

Vojtěch Juránek

Red Hat

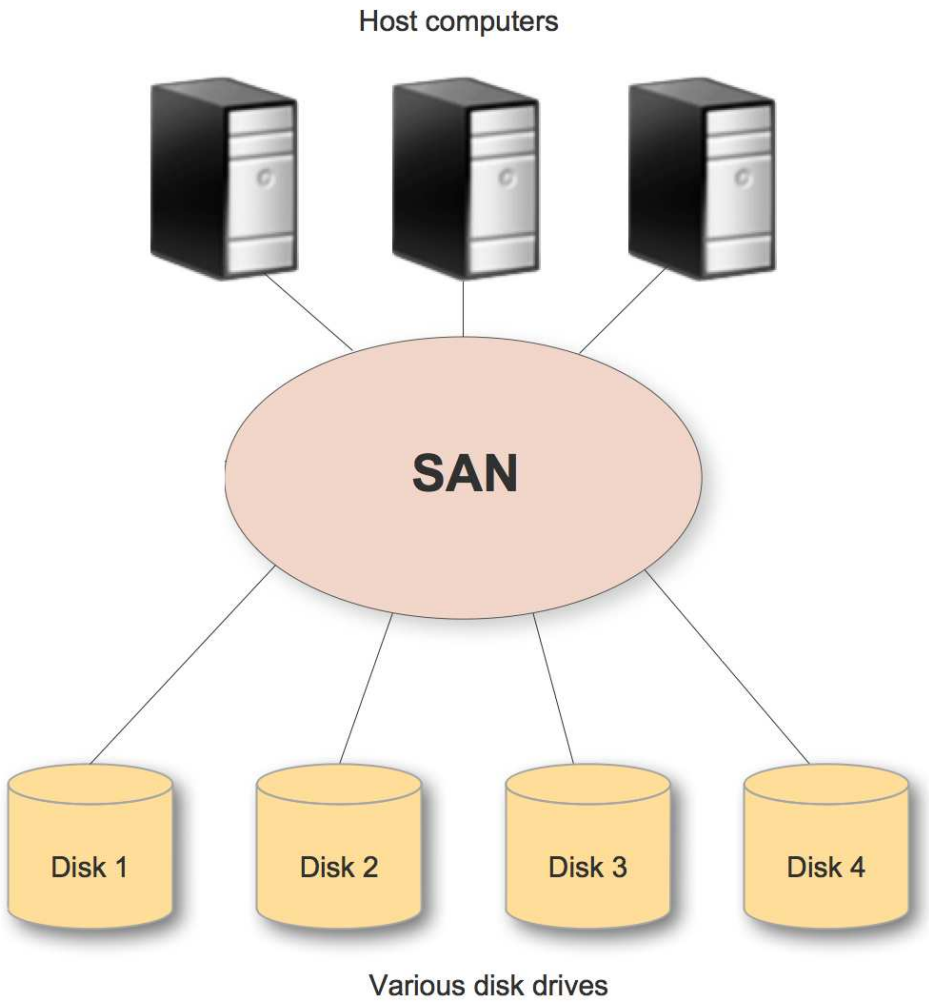
22. 10. 2019

Network attached storage



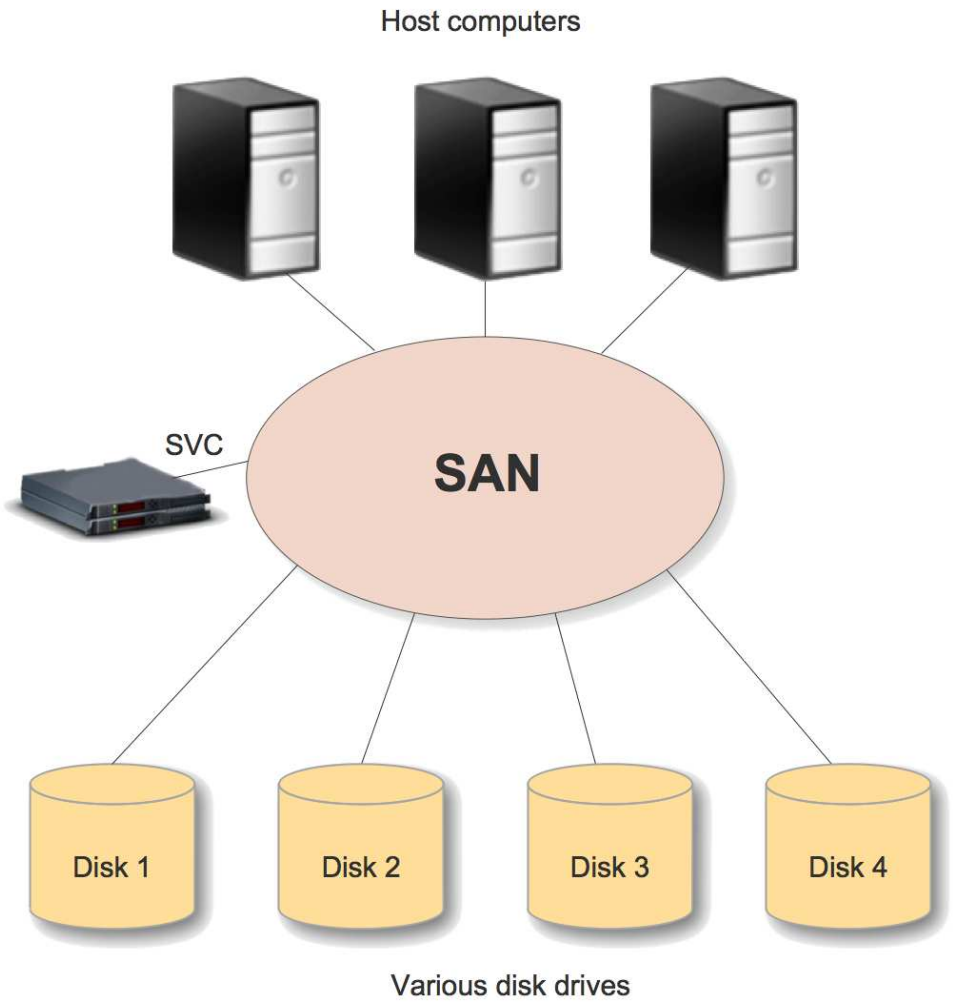
Source: `https://www.linuxjournal.com/content/high-availability-storage-ha-lvm`

Storage area network



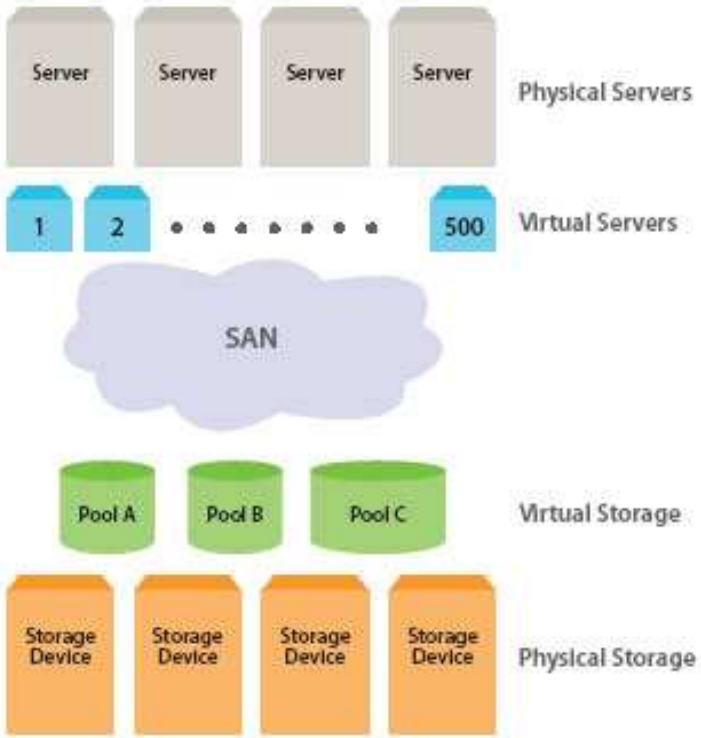
Source: `https://www.ibm.com/developerworks/community/wikis/`

Storage virtualization



Source: `https://www.ibm.com/developerworks/community/wikis/`

Storage virtualization



Source: `https://sharedstorage.wordpress.com/2017/01/03/storage-virtualization/`

Data consistency

- Dedicated (external) lock manager.
- What about co-locate locks with the data protected by the locks:
  - Accessing the locks together with accessing the data itself.

Refresh: Consensus algorithm

Algorithm has to satisfy 2 conditions:

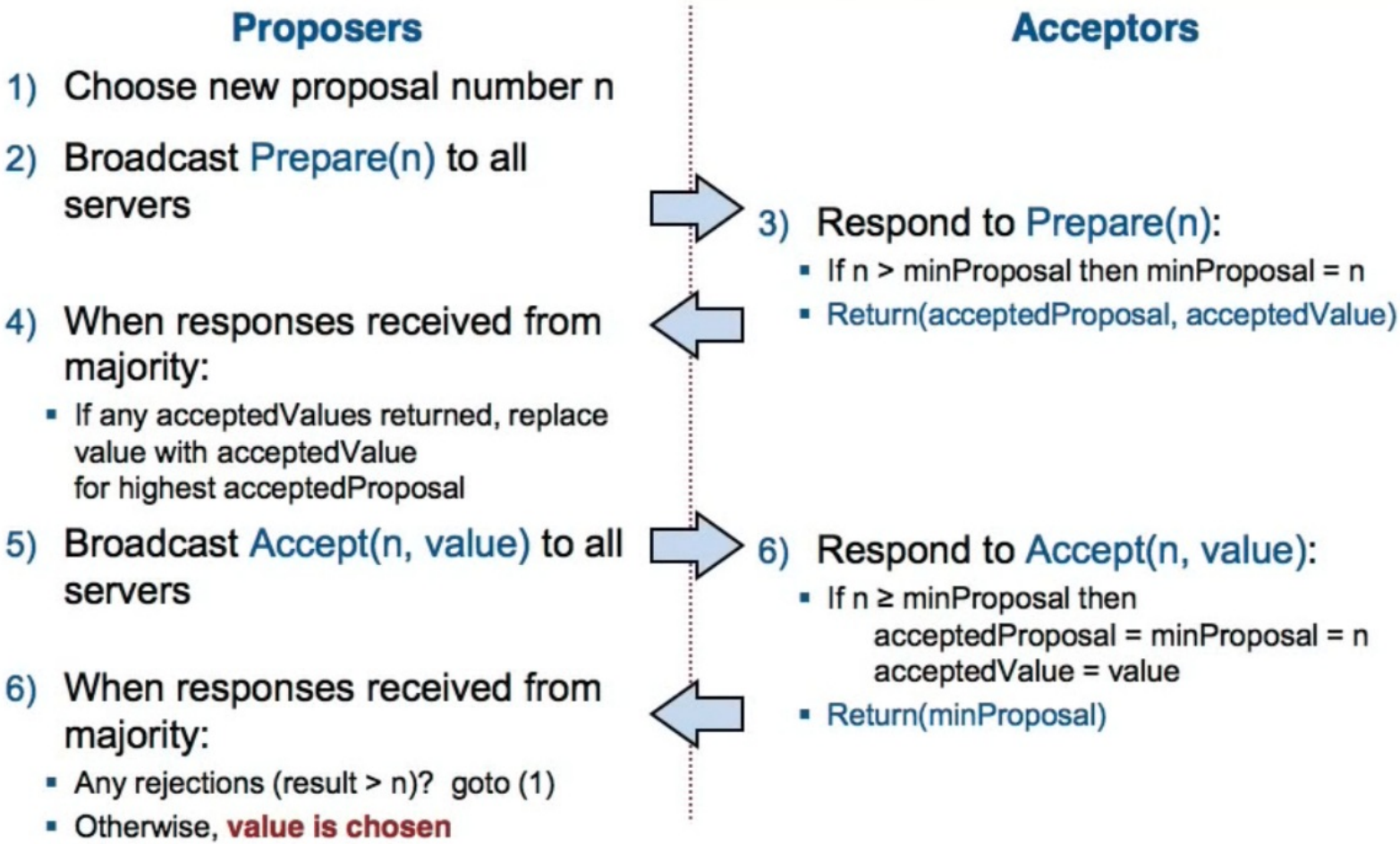
- safety (consistency)
- liveness (non-blocking)

Refresh: Classic Paxos

3-phase commit protocol called Synod:

- voting
- prepare to commit (pre-commit)
- commit

Refresh: Basic Paxos



Acceptors must record minProposal, acceptedProposal, and acceptedValue on stable storage (disk)

Source: `https://youtu.be/JEpsBg0AO6o?t=1050`

Disk Paxos

Disk Paxos

Eli Gafni[1] and Leslie Lamport[2]

[1] Computer Science Department, UCLA
[2] Compaq Systems Research Center

Abstract. We present an algorithm, called Disk Paxos, for implementing a reliable distributed system with a network of processors and disks. Like the original Paxos algorithm, Disk Paxos maintains consistency in the presence of arbitrary non-Byzantine faults. Progress can be guaranteed as long as a majority of the disks are available, even if all processors but one have failed.

Disk Paxos

Two main phases:

- proposing a value
- preparing the commit of a value

Disk Paxos

In whole algorithm we assume reading and writing are atomic operations.

- Each processor has an assigned block on each disk.
- Stores following structure `dblock[p]`:
  - `mbal` - ballot number
  - `bal` - largest ballot number for which `p` reached preparing commit phase
  - `inp` - value `p` tries to commit in ballot `bal`

Phase 1: proposing a value

For each disk `d`

- write `dblock[p]` to disk `[d][p]`
- read `dblock[p]` from `[d][q]` for all other processes `[q]`

For any disk `d` and process `q`:

- if `[d][q].mbal > dblock[p].mbal` → abort
- else if read majority of disks → phase finished

Phase 2: preparing the commit of a value

Choosing a value:

- `inp = dblock[q].inp`, where `dblock[q]` is block with largest `dblock[q].bal` among read blocks
- or value proposed by `p` is all read blocks haven't any value set

Phase 2: preparing the commit of a value

For each disk `d`

- write `dblock[p]` to disk `[d][p]`
- read `dblock[p]` from `[d][q]` for all other processes `[q]`

For any disk `d` and process `q`:

- if `[d][q].mbal > dblock[p].mbal` $\rightarrow$ abort and start phase 1 with higher ballot number
- else if read majority of disks $\rightarrow$ value committed, broadcast it

Broadcasting a value

If p learns that some other value was committed during the process $\rightarrow$ abort and output committed value.

Unsolved problems

- How to add processors.
- How to store location/mapping of processors on the disks.

Δ-Lease

Light-Weight Leases for Storage-Centric Coordination

Gregory Chockler*, Dahlia Malkhi†

April 22, 2004

Abstract

We propose light-weight *lease* primitives to leverage fault-tolerant coordination among clients accessing a shared storage infrastructure (such as network attached disks or storage servers). In our approach, leases are implemented from the very shared data that they protect. That is, there is no global lease manager, there is a lease per data item (e.g., a file, a directory, a disk partition, etc.) or a collection thereof. Our lease primitives are useful for facilitating exclusive access to data in systems satisfying certain timeliness constraints. In addition, they can be utilized as a building block for implementing dependable services resilient to timing failures. In particular, we show a simple lease based solution for fault-tolerant Consensus which is a benchmark distributed coordination problem.

Keywords: leases, file systems, mutual exclusion, consensus

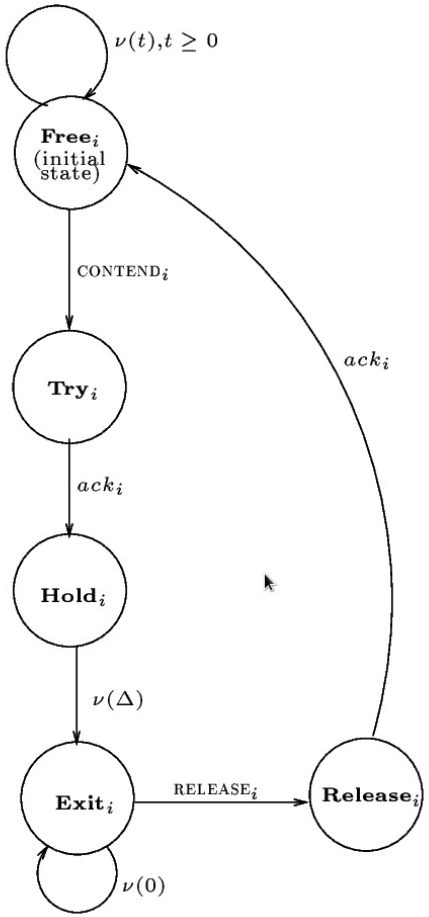
Known delay model

- In the following we will consider only known delay model (system is stable).
- Works also for eventually know delay model (exists global stabilization time after which system is stable).

The system is stable when:

- the process' clock drift with respect to the real time is bounded by a known constant ρ (in the following is assumed $\rho = 0$),
- the time it takes to correct process to complete its access to do some operation on shared memory object (typically read/write) is strictly less than a known bound δ.

Δ-Lease life cycle



Once acquired, lock is hold for time period Δ.

Δ-Lease safety and liveness

- Safety: At most one process is in the Hold state.
- Liveness:
  - If no process is in the Hold state, and some correct process is int he Try state, then at some later point some correct process enters the Hold state.
  - At any point in an execution, if a correct process `i` is in the Release state, then at some later point process `i` enters the Free state.

Δ-Lease Implementation

Shared:
$x \in TS_\perp$;
Local:
$x_1, x_2 \in TS_\perp$.

CONTEND:

```
(1)   x2 ← read(x);
(2)   do
(3)     if (x2 ≠ ⊥) then {
(4)       do
(5)         x1 ← x2;
(6)         delay(Δ + 5δ);
            /* Δ + 6δ for the ◊ND renewals */
(7)         x2 ← read(x);
(8)       until x1 = x2 ∨ x2 = ⊥;
          }
(9)     Generate a unique timestamp ts;
(10)    write(x, ts);
(11)    delay(2δ);
(12)    x2 ← read(x);
(13)  until x2 = ts;
(14)  return ack;
```

RELEASE:

```
      write(x, ⊥);
      return ack;
```

Δ-Lease

- Algorithm can be easily extended for eventually known delay model.
- We can add lease renewals.
- Leases can be used, besides locking, e.g. to a leader election.

sanlock

- https://pagure.io/sanlock
- Shared storage lock manager.
- Build on top of Disk Paxos and Δ-Leases.

sanlock Δ-Leases

- Δ-Leases are slow.
- For sanlock internal usage only.
- For keeping host ID
  - Prevents two hosts to have same ID.
  - Lease renewal is used also for checking is host is alive.

sanlock Disk Paxos leases

- Fast to acquire.
- Provided to application to as general purpose application leases.
- Host IDs are used as Paxos leases owners.

Summary

- For locking the data on a shared storage lock located with data can be better solution than dedicated lock manager.
- There are various algorithms for storage-centric locks, e.g. Disk Paxos, a modification of classical Paxos algorithm.
- Disk Paxos itself is not sufficient for full implementation of shared disk locks and has to be used with some other algorithm, e.g. Δ-Lease.
- Implementation of this approach can be found in sanlock project.

Questions?

. . . read Disk Paxos and Δ-Lease papers:

- E. Gafni, L. Lamport, Disk Paxos
- G. Chockler, D. Malkhi, Light-Weight Leases for Storage-Centric Coordination