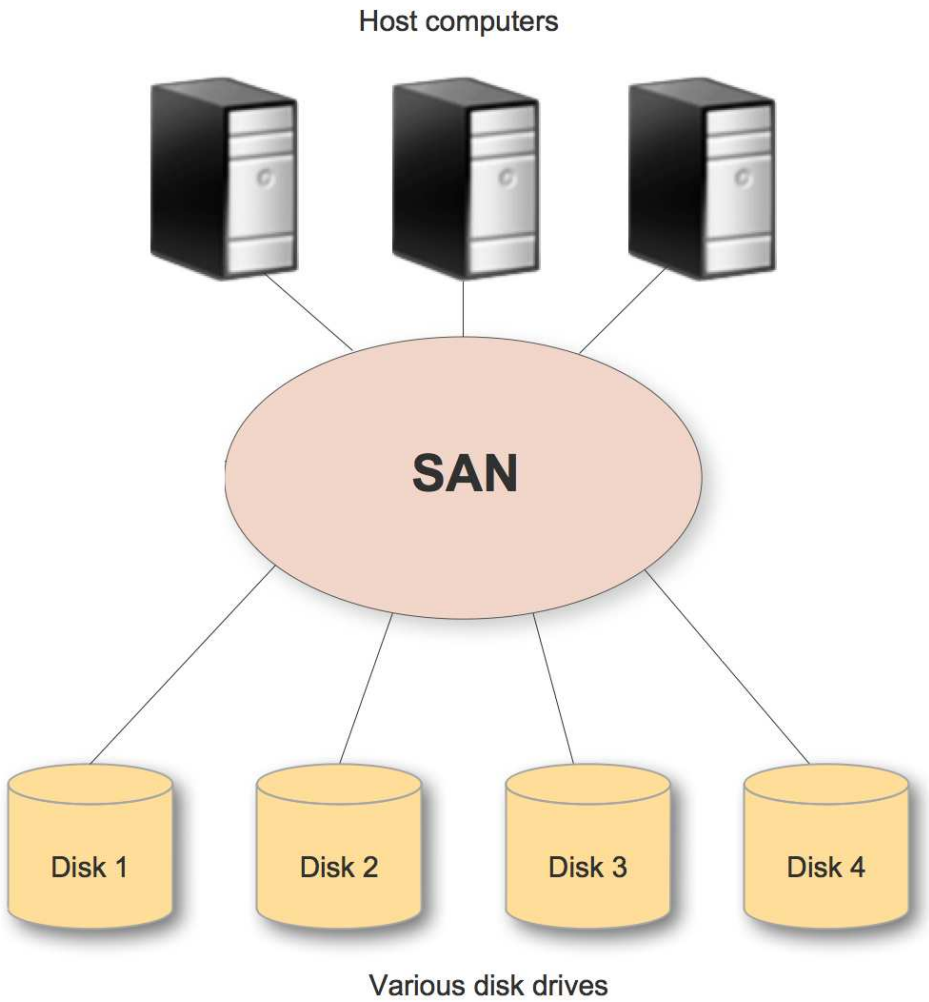
Protecting your resources with sanlock

Vojtěch Juránek

Red Hat

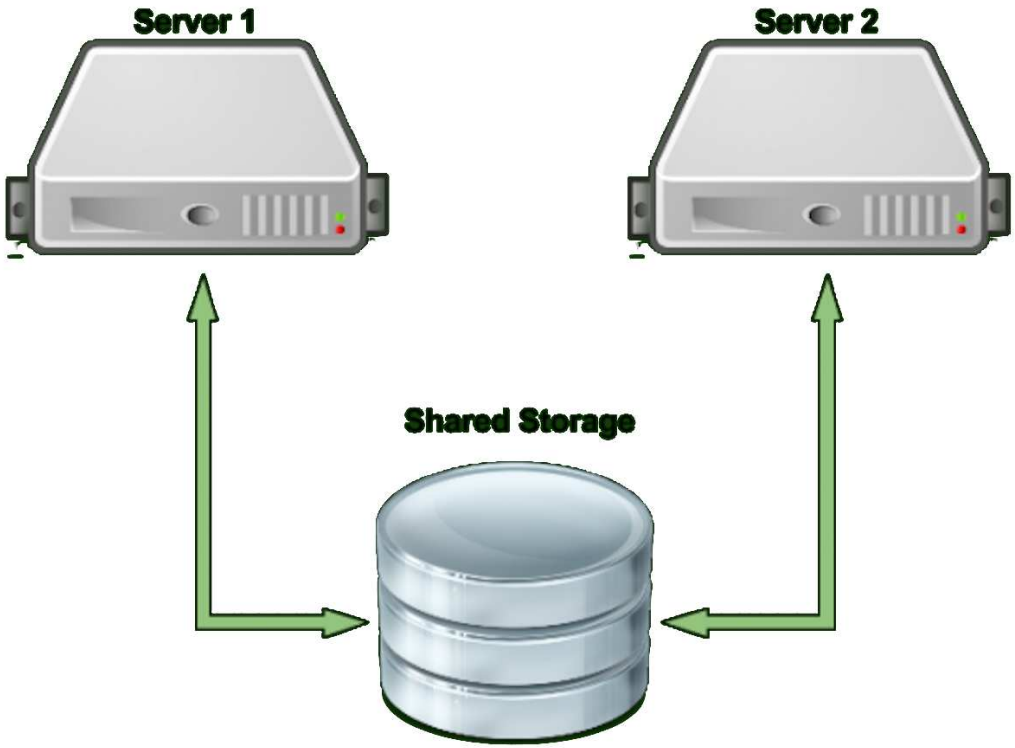
25. 1. 2020, DevConf.CZ, Brno

Storage area network



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

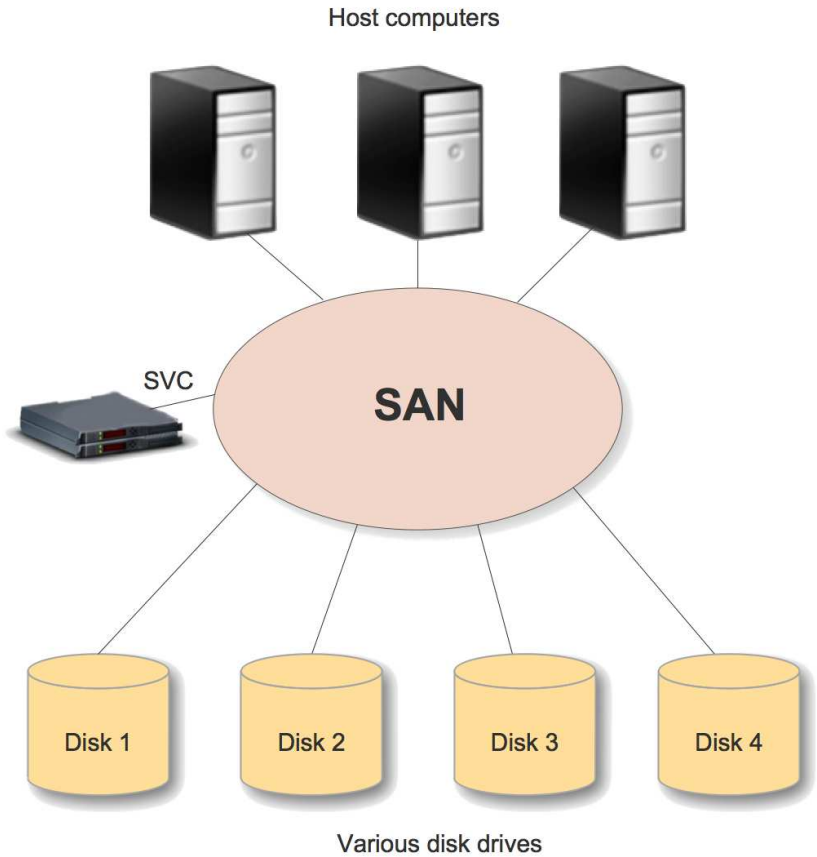
Data consistency



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

Data consistency

Dedicated (external) lock manager:



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

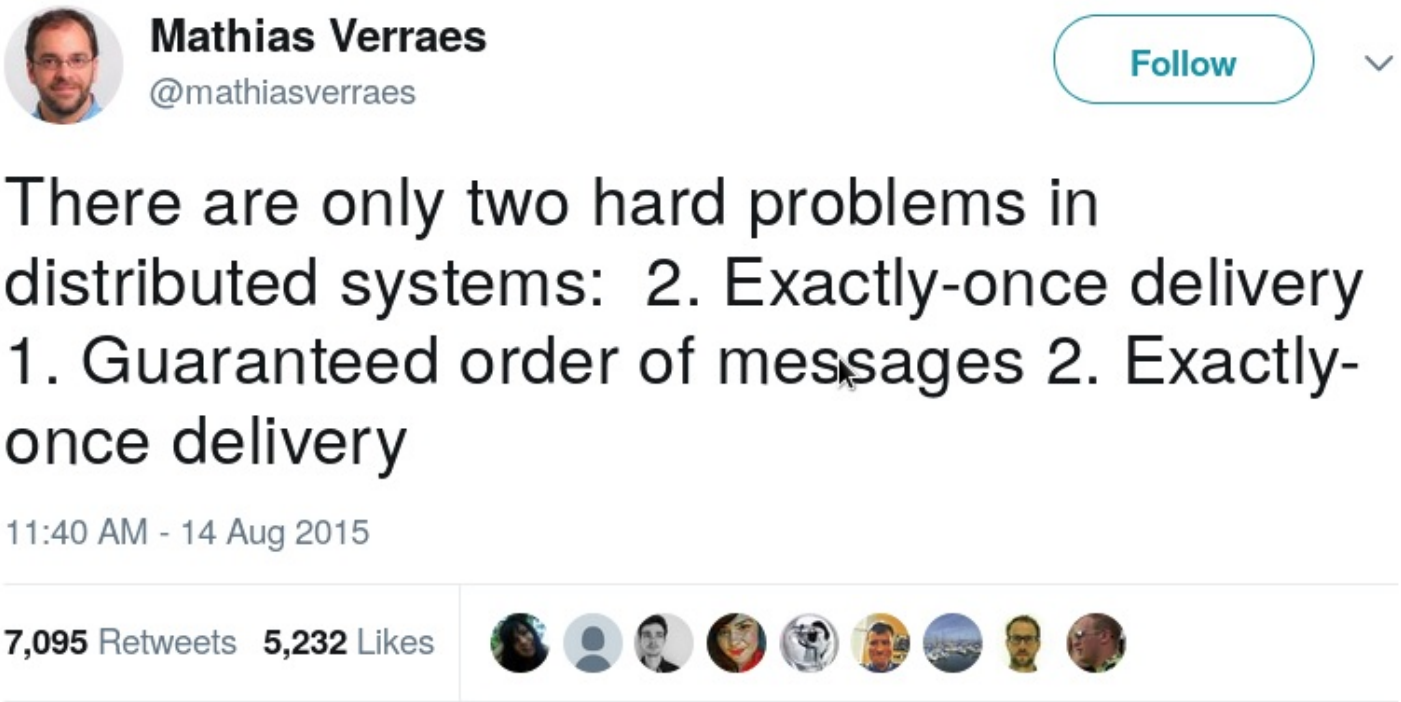
Data consistency

What about co-locate locks with the data and access the locks together with accessing the data itself.

sanlock

- Shared storage lock manager.
- Build on top of Disk Paxos and Δ-Leases algorithms.

Distributed consensus problem



Mathias Verraes
@mathiasverraes

Follow

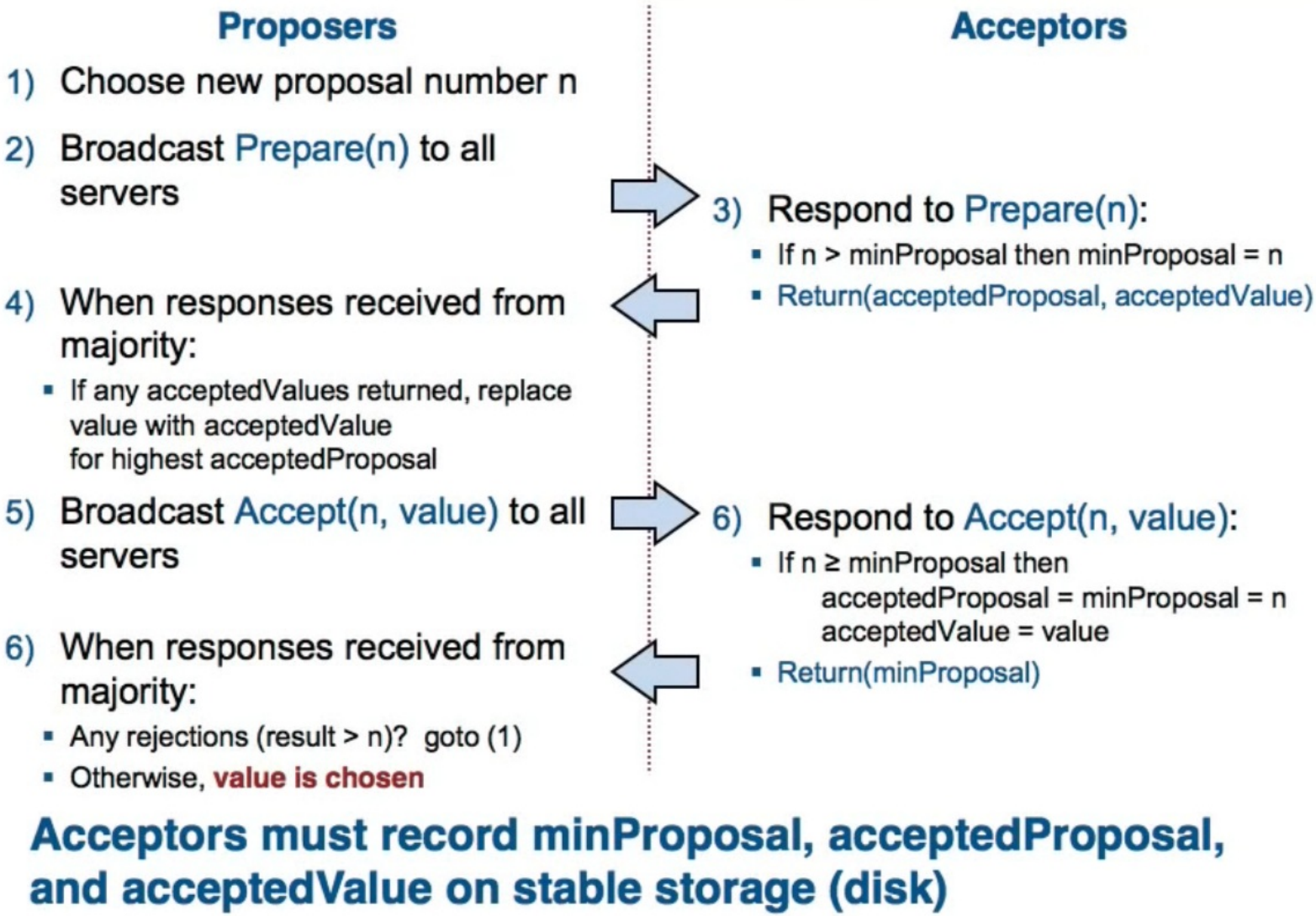
There are only two hard problems in distributed systems: 2. Exactly-once delivery 1. Guaranteed order of messages 2. Exactly-once delivery

11:40 AM - 14 Aug 2015

7,095 Retweets **5,232** Likes

Source: `https://twitter.com/mathiasverraes/status/632260618599403520`

Refresh: Basic Paxos



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

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

- Assume we have `m` disks `d` and `n` processors (computers/hosts) `p`.
- Each processor has an assigned block on each disk.
- Each processor stores on assigned block on each disk 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` → abort and start phase 1 with higher ballot number
- else if read majority of disks → value committed, broadcast it

Unsolved problems

- How to add processors.
- How to establish mapping of processors on the disks.

Light-Weight Leases for Storage-Centric Coordination

Gregory Chockler*, Dahlia Malkhi†

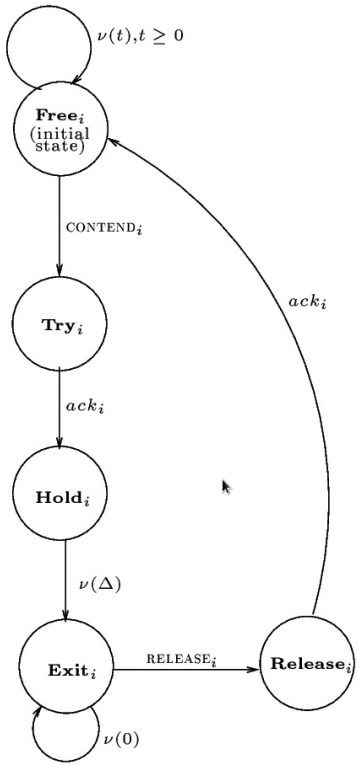
April 22, 2004

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

Δ-Lease life cycle



- The time it takes for 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 (known delay model).
- Once acquired, lock is hold for time period Δ.

Δ-Lease Implementation

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

CONTEND:

(1) $x_2 \leftarrow read(x)$;
(2) **do**
(3) **if** $(x_2 \neq \perp)$ **then** {
(4) **do**
(5) $x_1 \leftarrow x_2$;
(6) $delay(\Delta + 5\delta)$;
/* $\Delta + 6\delta$ for the ◊ND renewals */
(7) $x_2 \leftarrow read(x)$;
(8) **until** $x_1 = x_2 \vee x_2 = \perp$;
}
(9) Generate a unique timestamp ts;
(10) $write(x, ts)$;
(11) $delay(2\delta)$;
(12) $x_2 \leftarrow read(x)$;
(13) **until** $x_2 = ts$;
(14) return ack;

RELEASE:

$write(x, \perp)$;
return ack;

sanlock: big picture

- Δ-leases are used for acquiring/confirming unique ID for each host (only for sanlock internal usage)
- Paxos leases are used for acquiring locks for resources (meaning of the lock is determined by the application).
- Unique host IDs are use for Paxos leases for mapping host to its disk segment.

sanlock: Δ-Leases

- In sanlock terminology called *lockspaces*.
- Prevents two hosts to have same ID.
- Maximum number of hosts is 2000.
- Lockspace size is 1 MB (for block size 512B).
- Use options `-A` and `-Z` to change alignment and block size.

sanlock: Δ-Leases

- sanlock renews Δ-leases every 20 seconds by writing a new time stamp into appropriate Δ-lease block.
- Lease renewal is used also for checking if host is alive.

sanlock: Paxos leases

- Fast to acquire.
- In sanlock terminology called *resources*.
- Lease is acquired by the winner by writing its ballot into to the first block of the resource disk area.
- sanlock uses Δ-lease renewals also for Paxos leases renewal in given lockspace.

Using sanlock

Application which wants to use sanlock need to:

- initialize lockspace (once) on each host/disk == prepare disk space for sanlock,
- join the lockspace when it starts == acquire Δ-lease,
- acquire/release lease for a resource == acquire/release Paxos lease.
- leave the lockspace == release Δ-lease.

Summary

- sanlock combines Δ-lease and Disk Paxos algorithms in its own way.
- There are more parts, e.g. watchdog.
- Before changing anything, be sure you know what you are doing (e.g. before changing any timeouts, read at least `src/timeouts.h`).
- Understanding of Δ-lease and Disk Paxos algorithms should be a good start to understand rest of the sanlock.

Thank you!

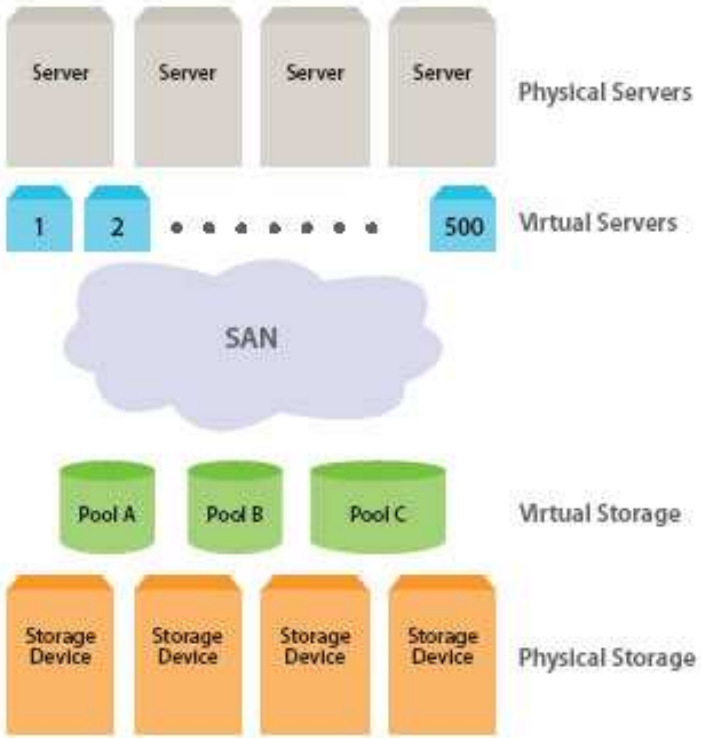
Questions?

. . . read sanlock source code, Disk Paxos and Δ-Lease papers :-)

- https://pagure.io/sanlock
- E. Gafni, L. Lamport, Disk Paxos
- G. Chockler, D. Malkhi, Light-Weight Leases for Storage-Centric Coordination

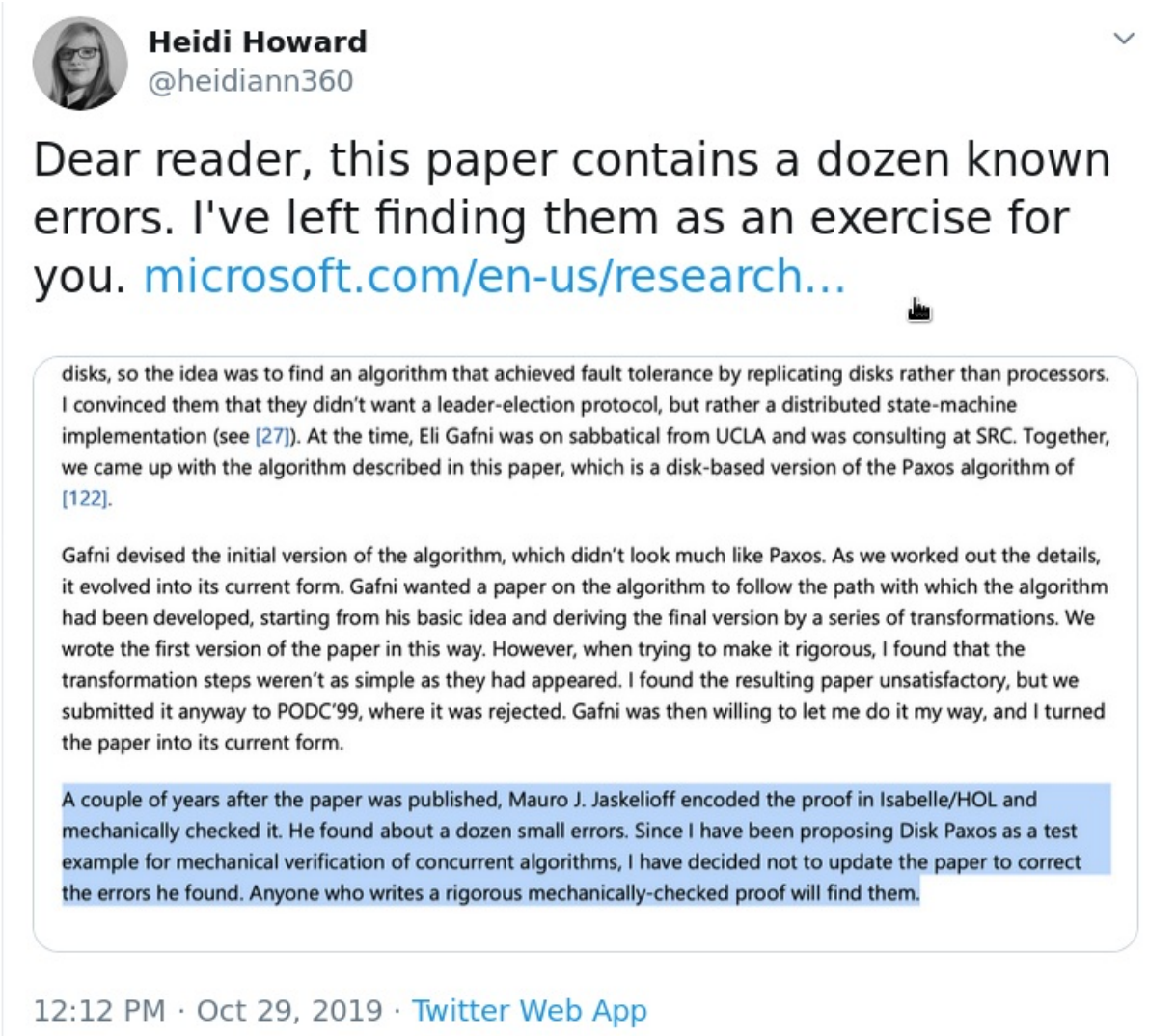
Backup slides

Data consistency example: Storage virtualization



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

Other problems



Heidi Howard
@heidiann360

Dear reader, this paper contains a dozen known errors. I've left finding them as an exercise for you. microsoft.com/en-us/research...

disks, so the idea was to find an algorithm that achieved fault tolerance by replicating disks rather than processors. I convinced them that they didn't want a leader-election protocol, but rather a distributed state-machine implementation (see [27]). At the time, Eli Gafni was on sabbatical from UCLA and was consulting at SRC. Together, we came up with the algorithm described in this paper, which is a disk-based version of the Paxos algorithm of [122].

Gafni devised the initial version of the algorithm, which didn't look much like Paxos. As we worked out the details, it evolved into its current form. Gafni wanted a paper on the algorithm to follow the path with which the algorithm had been developed, starting from his basic idea and deriving the final version by a series of transformations. We wrote the first version of the paper in this way. However, when trying to make it rigorous, I found that the transformation steps weren't as simple as they had appeared. I found the resulting paper unsatisfactory, but we submitted it anyway to PODC'99, where it was rejected. Gafni was then willing to let me do it my way, and I turned the paper into its current form.

A couple of years after the paper was published, Mauro J. Jaskelioff encoded the proof in Isabelle/HOL and mechanically checked it. He found about a dozen small errors. Since I have been proposing Disk Paxos as a test example for mechanical verification of concurrent algorithms, I have decided not to update the paper to correct the errors he found. Anyone who writes a rigorous mechanically-checked proof will find them.

12:12 PM · Oct 29, 2019 · Twitter Web App

Source: `https://twitter.com/heidiann360/status/1189137985830297600`