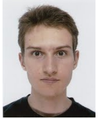


Alex Auvolat
Member of Deuxfleurs, lead developer of Garage

Garage
A self-hosted alternative to S3 for object storage

Deuxfleurs
A non-profit self-hosting collective,
member of the CHATONS network

Stable vs Resilient

Building a "stable" system:

Enterprise-grade systems typically employ:

- RAID
- Redundant power grid + UPS
- Redundant Internet connections
- Low-latency links
- ...

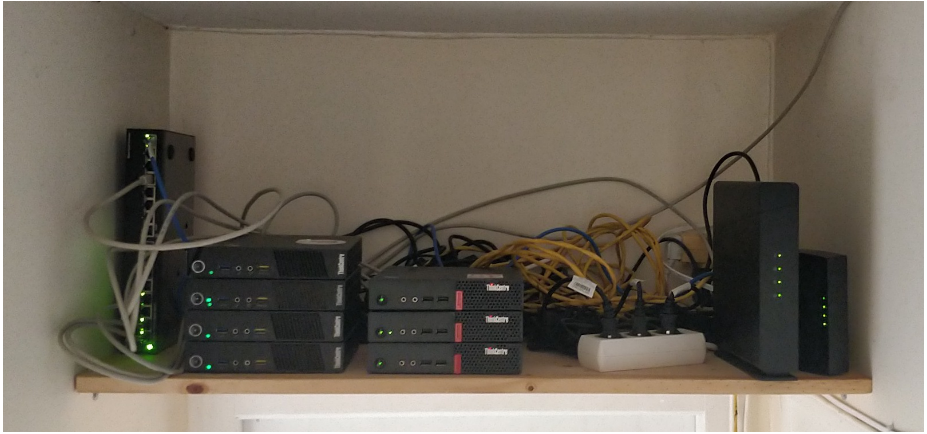
→ costly, only worth at DC scale
→ still risk of DC-level incident...

Building a resilient system:

An alternative, cheaper way:

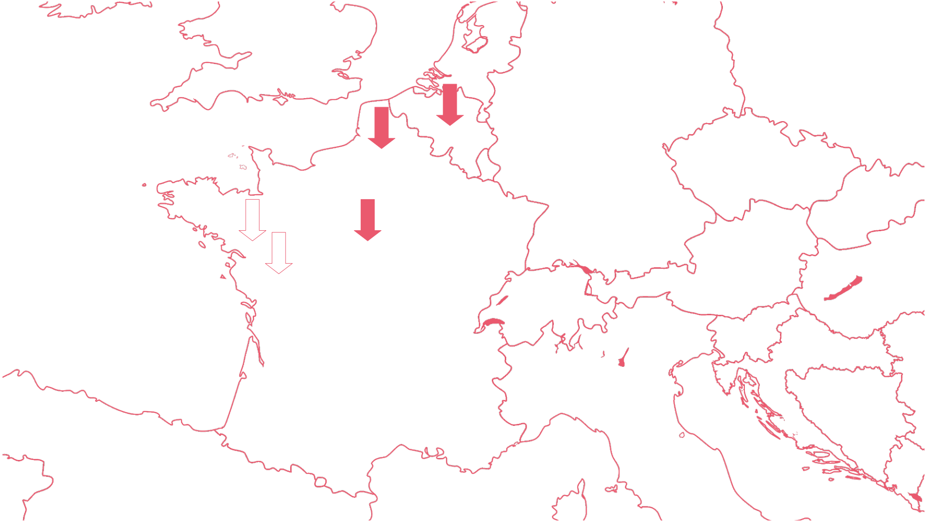
- Commodity hardware
  (e.g. old desktop PCs)
- Commodity Internet
  (e.g. FTTB, FTTH) and power grid
- **Geographical redundancy**
  (multi-site replication)

Example: our infrastructure at Deuxfleurs



Example: our infrastructure at Deuxfleurs

Example: our infrastructure at Deuxfleurs



Object storage: simpler than file systems

Only two operations:

- Put an object at a key
- Retrieve an object from its key

(and a few others)

Sufficient for many applications!

Amazon S3

MINIO

Garage

The data model of object storage

Object storage is basically a key-value store:

Key: file path + name	**Value: file data + metadata**

`index.html`	`Content-Type: text/html; charset=utf-8`
	`Content-Length: 24929`
	`<binary blob>`

`img/logo.svg`	`Content-Type: text/svg+xml`
	`Content-Length: 13429`
	`<binary blob>`

`download/index.html`	`Content-Type: text/html; charset=utf-8`
	`Content-Length: 26563`
	`<binary blob>`

Implementation: consensus vs weak consistency

Consensus-based systems:

- **Leader-based:** a leader is elected to coordinate all reads and writes
- Allows for **sequential reasoning**: program as if running on a single machine
- Serializability is one of the **strongest consistency guarantees**
- **Costly**, the leader is a bottleneck; leader elections on failure take time

Implementation: consensus vs weak consistency

Consensus-based systems:

- **Leader-based:** a leader is elected to coordinate all reads and writes
- Allows for **sequential reasoning**: program as if running on a single machine
- Serializability is one of the **strongest consistency guarantees**
- **Costly**, the leader is a bottleneck; leader elections on failure take time

Weakly consistent systems:

- **Nodes are equivalent**, any node can originate a read or write operation
- **Operations must be independent**, conflicts are resolved after the fact
- Strongest achievable consistency: **read-after-write consistency** (using quorums)
- **Fast**, no single bottleneck; works transparently with offline nodes

Why avoid consensus?

Consensus can be implemented reasonably well in practice, so why avoid it?

- **Software complexity:** RAFT and PAXOS are complex beasts;
  harder to prove, harder to reason about
- **Performance issues:**
  - Taking a decision may take an **arbitrary number of steps** (in adverse scenarios)
  - The leader is a **bottleneck** for all requests;
    even in leaderless approaches, **all nodes must process all operations in order**
  - Particularly **sensitive to higher latency** between nodes

Objective: the right level of consistency for Garage

Constraints: slow network (geographical distance), node unavailability/crashes

Objective: maximize availability, maintain an *appropriate level of consistency*

Objective: the right level of consistency for Garage

Constraints: slow network (geographical distance), node unavailability/crashes
Objective: maximize availability, maintain an *appropriate level of consistency*

1. **Weak consistency for most things**

   Example: `PutObject`

   If two clients write the same object at the same time, one of the two is implicitly overwritten. No need to coordinate, use a *last-writer-wins register*.

Objective: the right level of consistency for Garage

Constraints: slow network (geographical distance), node unavailability/crashes
Objective: maximize availability, maintain an *appropriate level of consistency*

1. **Weak consistency for most things**

   Example: `PutObject`

   If two clients write the same object at the same time, one of the two is implicitly overwritten. No need to coordinate, use a *last-writer-wins register*.

2. **Stronger consistency only when necessary**

   Example: `CreateBucket`

   A bucket is a reserved name in a shared namespace, two clients should be prevented from both creating the same bucket (*mutual exclusion*).

The possibility of *leaderless consensus*

Currently, Garage *only has weak consistency*. Is fast, but `CreateBucket` is broken!

The possibility of *leaderless consensus*

Currently, Garage *only has weak consistency*. Is fast, but `CreateBucket` is broken!

Leaderless consensus (Antoniadis et al., 2023) alleviates issues with RAFT and PAXOS:

- **No leader.** All nodes participate equally at each time step, and different nodes can be unavailable at different times without issues.

  → better tolerance to the high latency (remove bottleneck issue)
  → tolerates crash transparently

- **Simpler formalization.** The algorithm is very simple to express and to analyze in mathematical terms.

The possibility of *leaderless consensus*

Currently, Garage *only has weak consistency*. Is fast, but `CreateBucket` is broken!

Leaderless consensus (Antoniadis et al., 2023) alleviates issues with RAFT and PAXOS:

- **No leader.** All nodes participate equally at each time step, and different nodes can be unavailable at different times without issues.

  → better tolerance to the high latency (remove bottleneck issue)
  → tolerates crash transparently

- **Simpler formalization.** The algorithm is very simple to express and to analyze in mathematical terms.

One of the possible subjects for this PhD:
→ *integration of leaderless consensus in Garage* + testing + perf eval, etc.

Garage

`https://garagehq.deuxfleurs.fr/`
`mailto:garagehq@deuxfleurs.fr`
`#garage:deuxfleurs.fr` on Matrix