RainbowFS

Modular Consistency and Co-designed Massive File system

M. Shapiro, UPMC & Inria, Paris
B. King, Scality SA, Paris
V. Quéma, CNRS-LIG, Grenoble
P. Sutra, Télécom ParisSud, Évry
S. Monnet, U. Savoie-Mont Blanc, Annecy

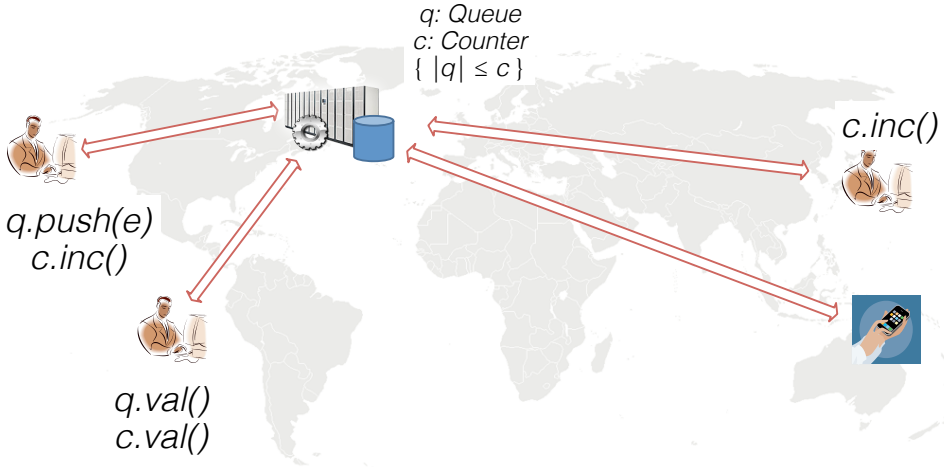
Background

Distributed application ⟹ shared data

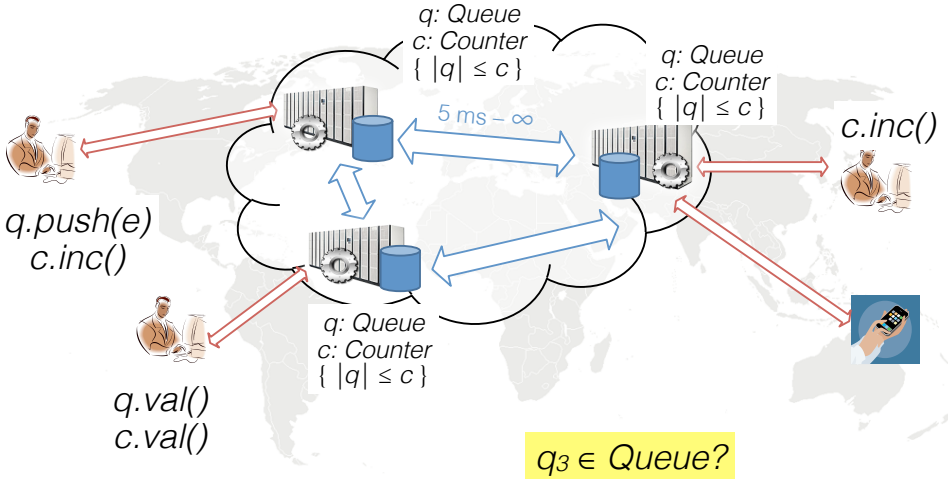
1. Large scale (cloud) ⟹ data replication ⟹ consistency issues
   - Strong : dependable, not available, inefficient.
   - Available : parallel, anomalies.
   - Pre-defined models
2. Complex : configuration, analysis, control, decomposition
3. System, data access API

Theory of replication and consistency, tools

Shared data

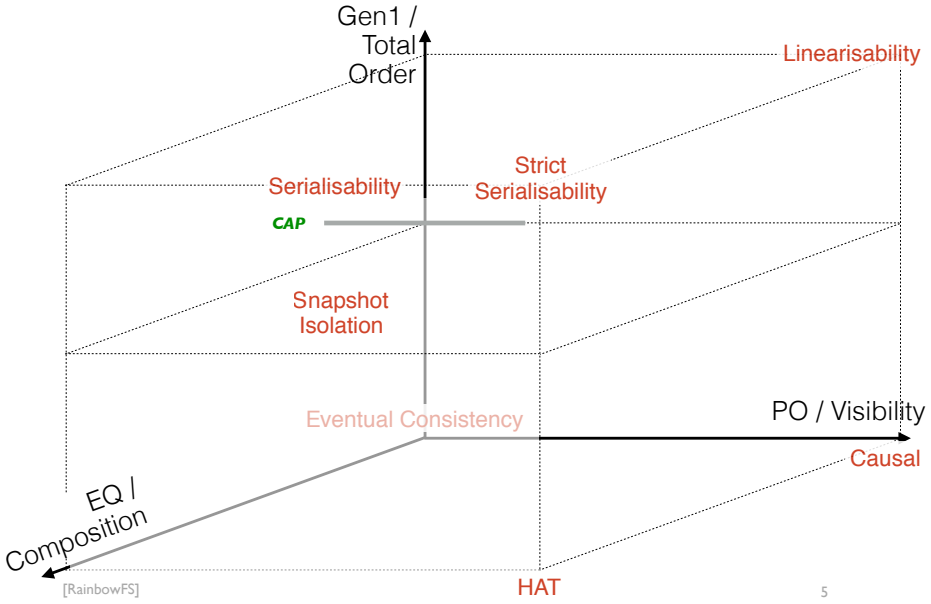


Geo-replicated sharing



$q_3 \in Queue?$
$q_1 = q_2$?
$|q_1| \leq c_4$?

Three dimensions



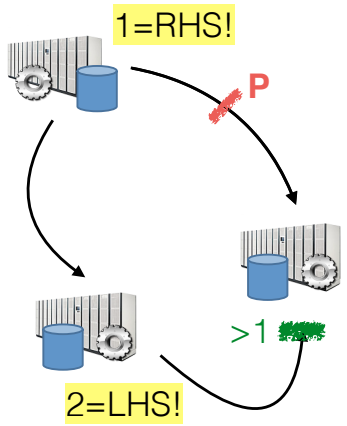
Relative order pattern

mkdir before *creat*

Relative-order invariant pattern:

- "Directory references valid file"
- *x* valid ∧ *x* points to *y* ⟹ *y* valid
- Pattern RHS!; LHS!
- Make visible in same order

AP-compatible: Causal Consistency



Joint update pattern

write updates content, metadata

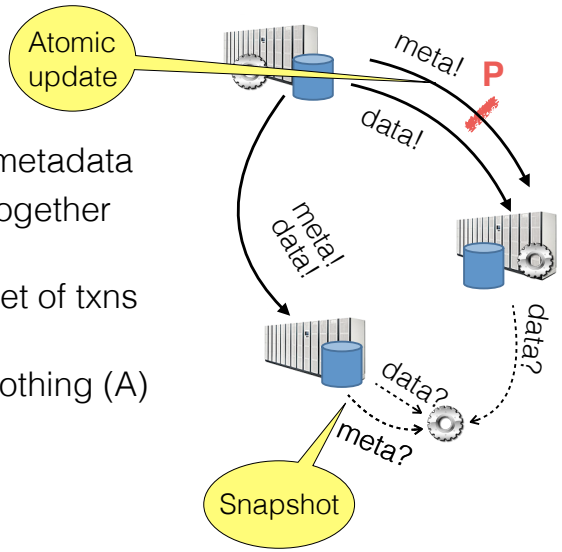
Transmit joint updates together

- write-atomic

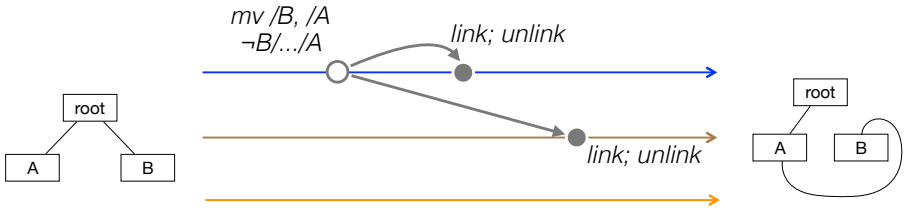
+ Read from common set of txns

- snapshot property

AP-compatible: All-or-Nothing (A)

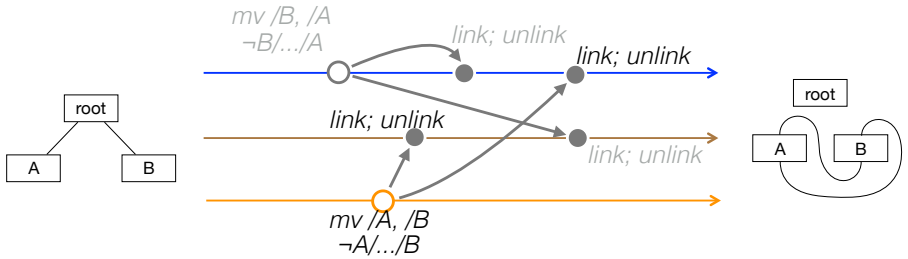


CAP-sensitive pattern



mv (node, ddir) { // tree
  if ¬ *ancestor (node, ddir)* // precond: at source
    link (ddir, node) // at every …
    unlink (sdir, node) // … replica
} // tree

CAP-sensitive pattern



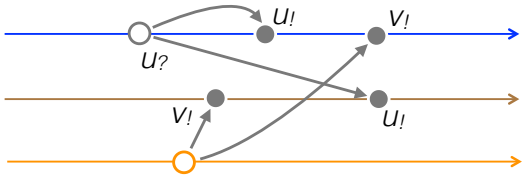

```
mv (node, ddir) { // tree
  if ¬ ancestor (node, ddir)    // precond: at source
    link (ddir, node)           // at every …
    unlink (sdir, node)         // … replica
} // tree
```

CISE Static Analysis

Precondition *not stable* w.r.t. concurrent *mv*

- Forbid concurrency? Synchro, CP.
- Or remove invariant? AP, degraded semantics

A data model for AP



Concurrent, asynchronous updates

- Standard register model: assignments ⟹ CP
- AP ⟹ concurrent updates merged

CRDT: register, counter, set, map, sequence

- Extends sequential type
- Encapsulates convergent merge

geoFS

Scality—UPMC

Tao Thanh Vinh

Posix-like replicated file system

- Available under Partition
- CRDTs: merge concurrent updates
- || *write* same file: merge or rename
- *delete* || *write*: file path survives
- *mv* || *mv* ⟶ copy; delete

mv || *mv* ⟹ unstable precondition

- Either CP or anomalous
- CP: minimal synch footprint

Mahsa Najafzadeh

RainbowFS

ANR project 2017–2021

UPMC + LIG + TSP + Scality + USMB

1. Application/system co-design
2. Modular replication
3. Multi-consistency file system

1. Application / consistency co-design

Just-Right Consistency:

- Most efficient consistency…
- …under which *my* application is correct

Static and dynamic verification

File system:

- Maintain tree invariant; others?
- Asynchronous: *mkdir, rmdir, creat, rm,* etc.
- Design options: *write* || *write, write* || *rm*
- Synchronised: *mv*

2. Modular geo-replication

From application skeleton to running system

Tools for large-scale deployment, monitoring, analysis, diagnosis

- Check correctness & efficiency
- Stress test
- Diagnose root cause

Protocol building blocks, compose for application

- 3D model, variants/strength
- Modular fault tolerance (Abstract)

3. Geo-replicated massive file system

Build a file system:

- Posix API
- Multi-consistency, tailor to application
- Scales to O(petabyte)
- Apply tools & methods from Tasks 1 & 2

Challenges:

- Geo-distributed, elastic
- Massive performance, partial replication
- Consistency, security, fault tolerance
- Layer above object store

Creative Commons Attribution-ShareAlike 4.0 Intl. License

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material

for any purpose, even commercially, under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.