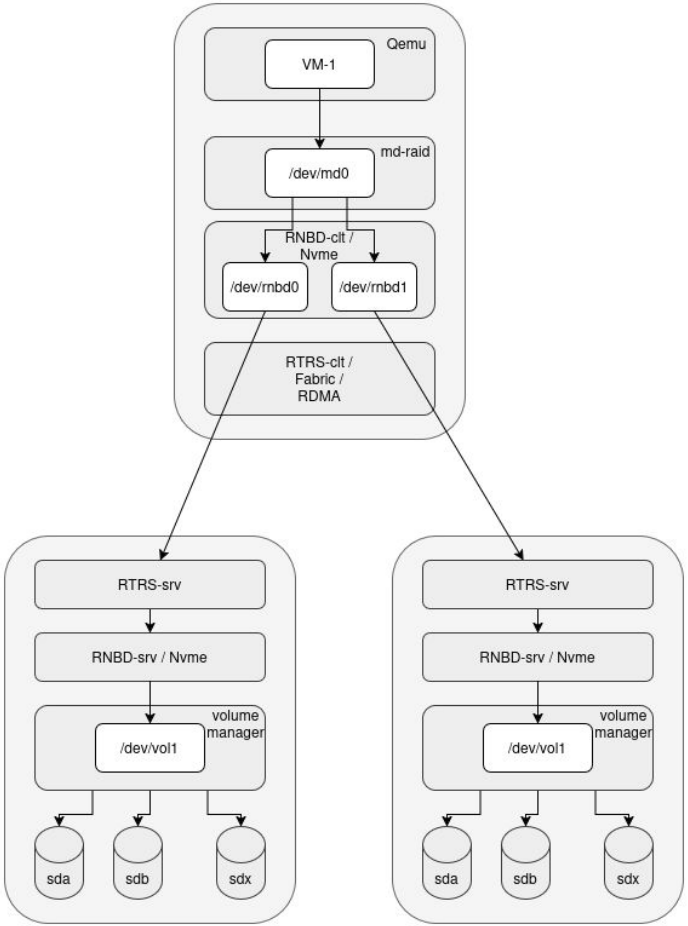
RMR & BRMR

Block level active-active replication over RDMA

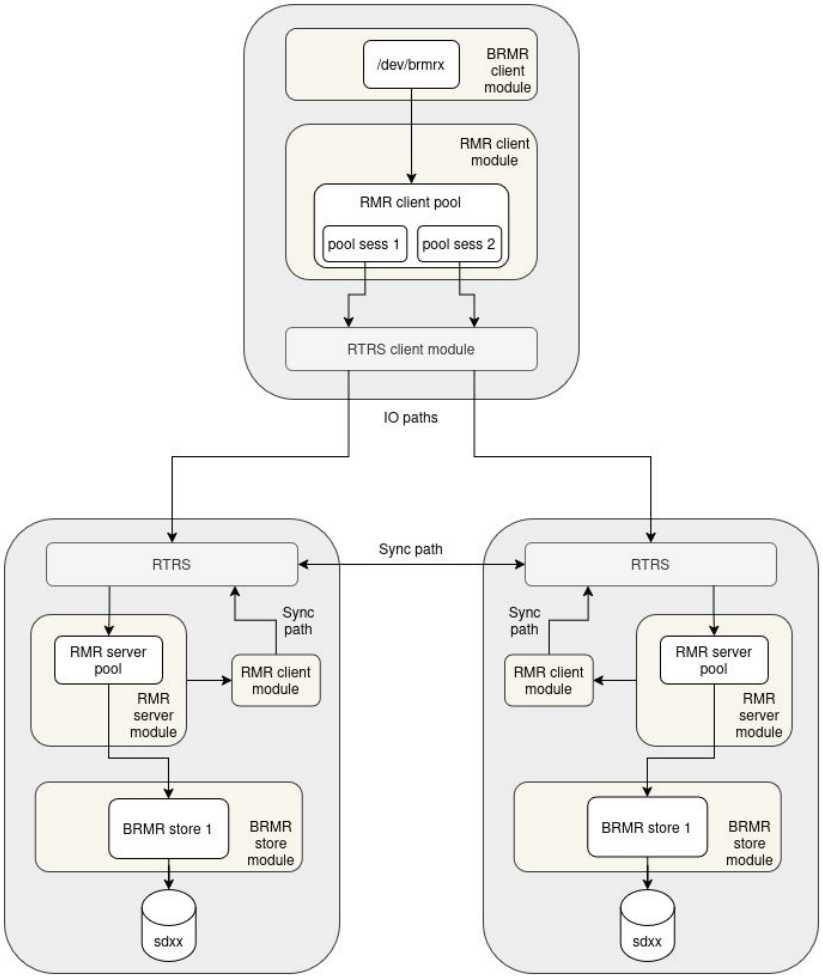
Haris Iqbal, Jia Li
IONOS

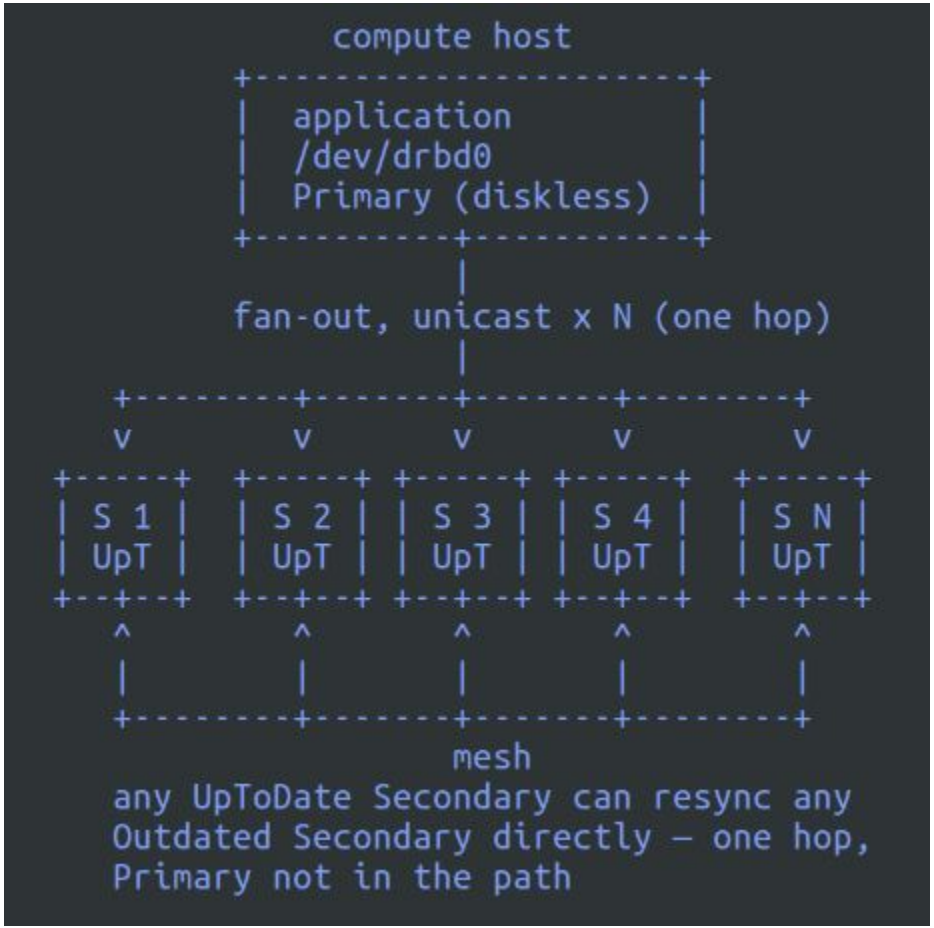


Block level replication

Overview

- Block level active-active replication.
- Single hop IO, single hop sync.
- Uses dirty maps to track failed IOs in case of storage node failures.
- Uses list of last IO tracking to achieve data consistency in case of client failure.

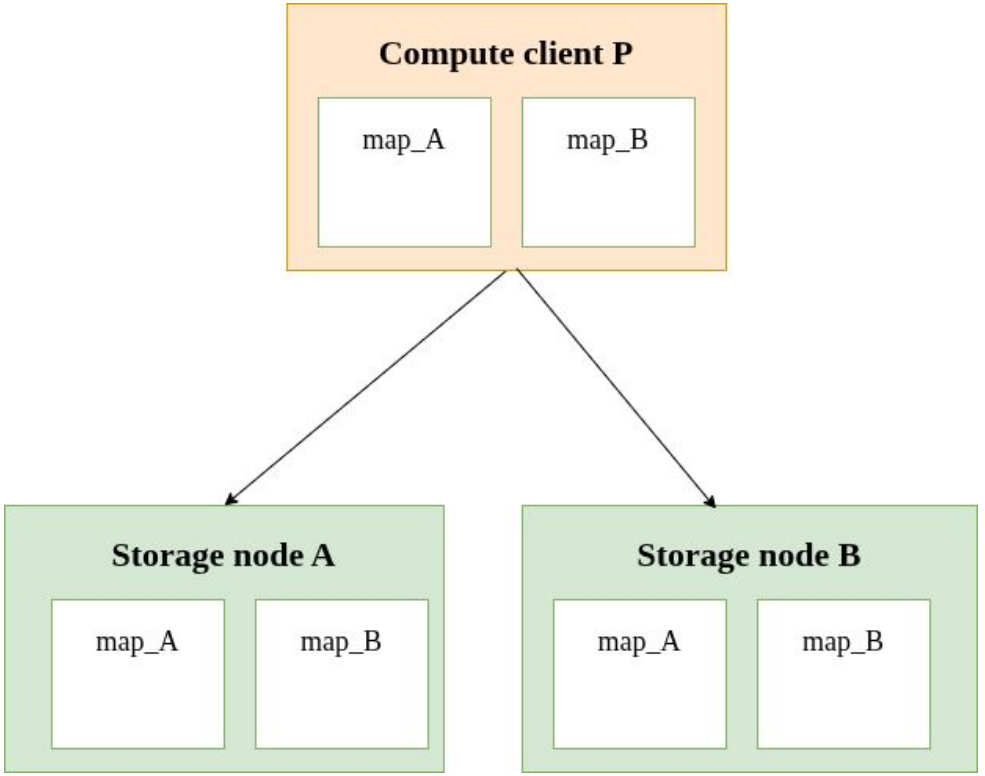



compute host
application
/dev/drbd0
Primary (diskless)
fan-out, unicast x N (one hop)
S 1
UpT
S 2
UpT
S 3
UpT
S 4
UpT
S N
UpT
mesh
any UpToDate Secondary can resync any
Outdated Secondary directly – one hop,
Primary not in the path

Discussions

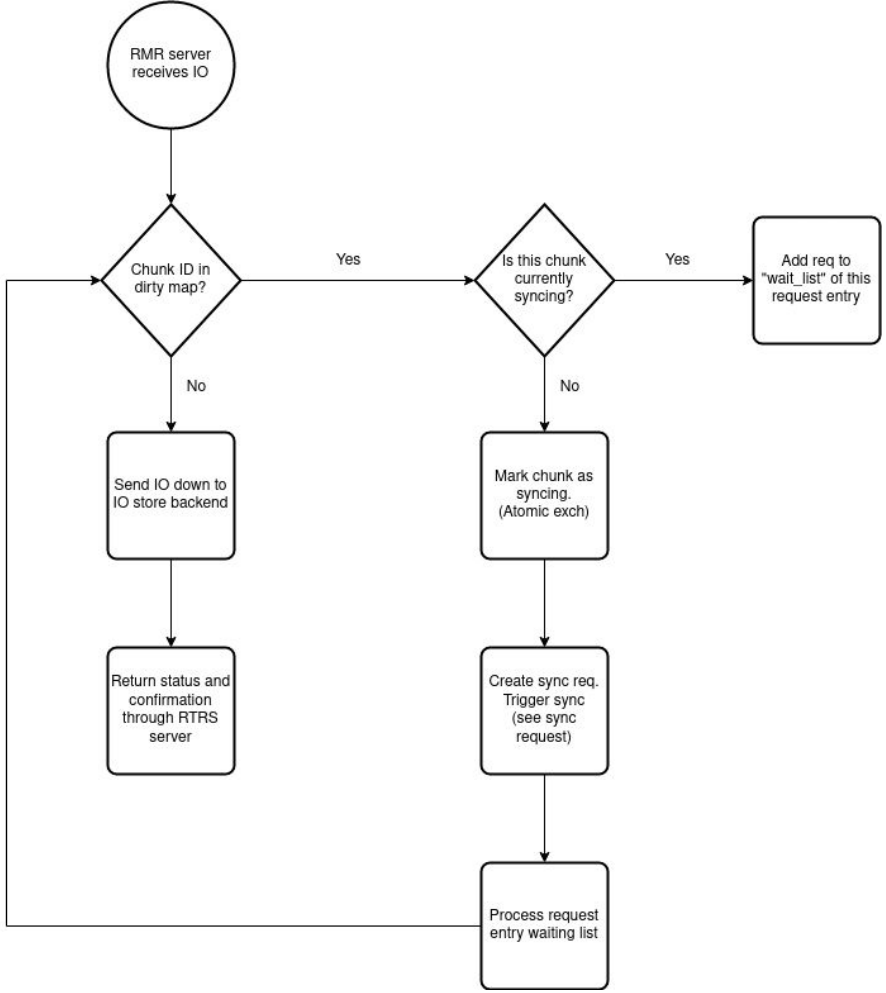
- Why new kernel modules?
  - We wanted to keep the reliable multicast transport layer (RMR) independent for future use cases.
    - A KV store, an S3 object store, etc.
- Active-active VS active-passive
- Diskless DRBD 9 Primary + mesh — has anyone run it at scale?

- **Dirty maps** tracks which chunks need synchronization through a bitmap buffer where each bit represents a chunk number (offset/chunk_size).
- RMR uses a two-level page table where leaf pages store the actual dirty map.
- **Map update** is updating the maps on a storage node to the most up-to-date ones.
- During updating the maps, IOs from compute client are frozen to make sure the maps is unchanged.
- When the process starts from the client, it picks a normal node to send maps to the dedicated node.



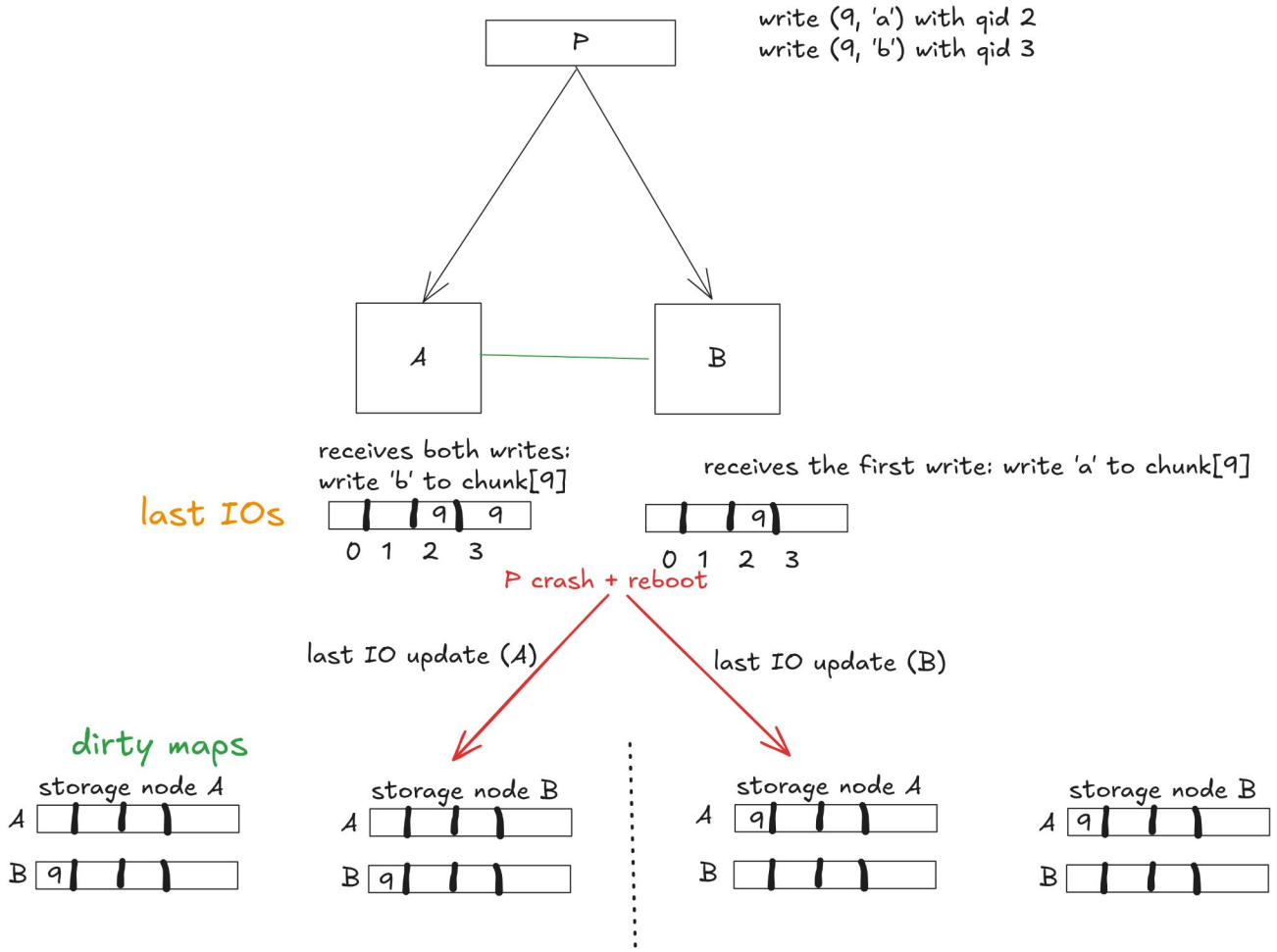
Map update

- Client-side
  - Read requests are sent to sessions in NORMAL state, in a round robin manner.
  - Write requests are completed if it completes for at least one session. Although the RMR client waits to get responses for all sessions, success or failure.
  - For failed write requests, a map_add is sent to all storage nodes in NORMAL state.
- Server-side: Check if the storage server is healthy to service IOs.
- Sync request: On server side, a sync thread in the background synchronizes the dirty data. It also locks the chunks while data is syncing, so that all incoming IO goes to the “wait list”.



Data path: read, write, sync

- The **last IOs** track the last queue_depth number of IOs completed by a storage node.
- Each successful write IO records the chunk index at index qid. So at any moment, every storage node holds a record of the most recent queue_depth chunk writes it has acknowledged.
- **Last IO update** converts those per-node last-IO records into dirty-map entries, after a compute client crash, so that any chunk whose write might have been partial is correctly marked dirty.



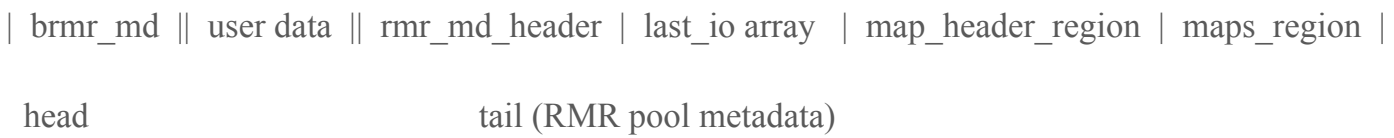
Last IO update

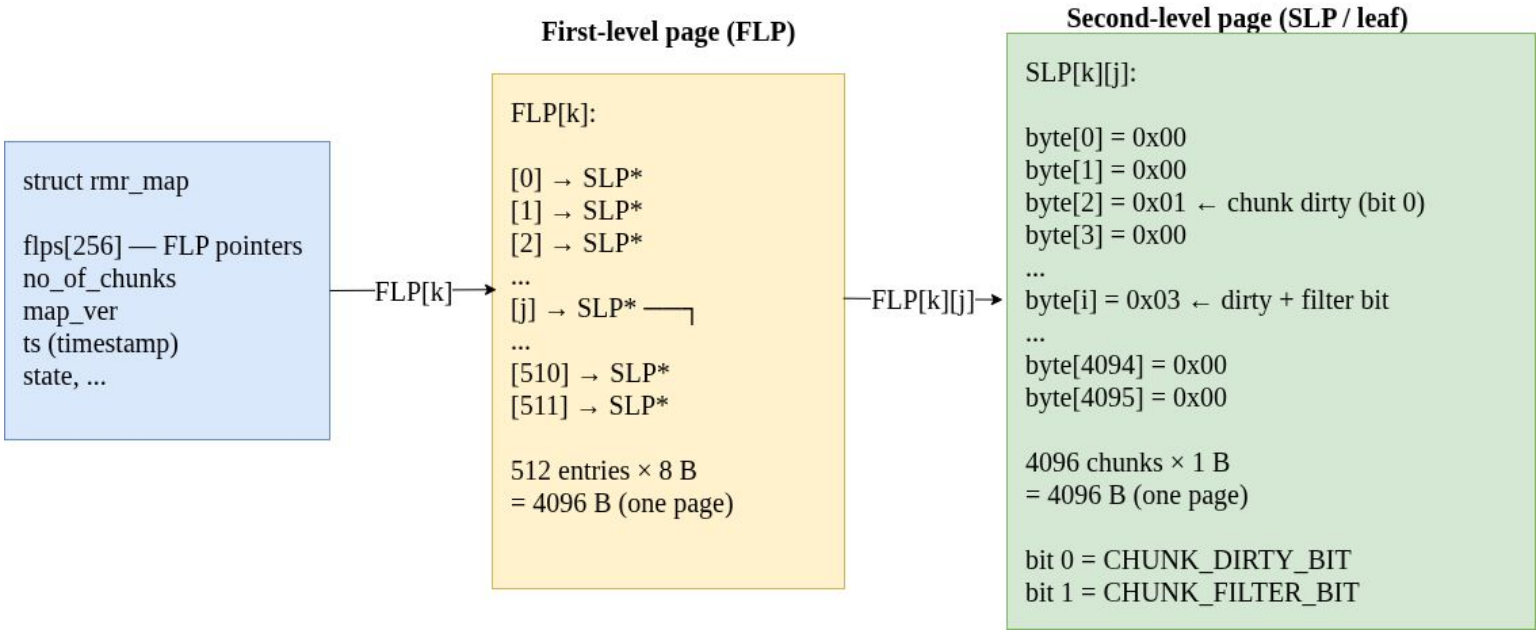
Failure recovery

1. Network/IO failure: Map update
2. Storage node crashes: Map update
3. Compute client crashes: Last IO update
4. All nodes crash: Metadata persistence + last IO update

Metadata persistence

- The layout of the RMR pool metadata on the disk is as the following:
  - BRMR metadata sits at the head — owned by the brmr-store layer.
  - User data in the middle — the addressable block range exposed to BRMR clients.
  - RMR pool metadata at the tail
- On-demand sync: Each metadata region has a dirty bit. Whoever changes a region sets its bit and arms a delayed worker. The worker wakes once, flushes only the flagged regions, then sleeps. No producer means no IO.





Map internal: 2-level page table

Links

- https://github.com/ionos-cloud/RMR
- https://ionos-cloud.github.io/rmr.io/index.html#