Practices on Lustre File-level RAID

Qi Chen

chenqi.jn@gmail.com

Jiangnan Institute of Computing Technology

Agenda

- Background
  - motivations
  - practices on client-driven file-level RAID
- Server-driven file-level RAID
  - fundamentals and key ideas
  - overview of the architecture
  - advantages and limitations
- Current state of development
- Future work

Motivations

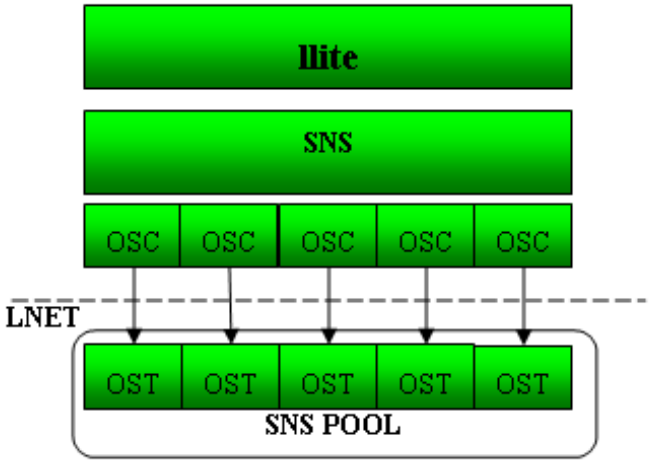
- Two challenges of distributed file system
  - as system scales, node failure is becoming more frequent
  - as disk capacity increase, Hardware RAID is becoming more expensive
- Solutions
  - high availability
    - heartbeat/zookeeper
    - failover
  - high reliability
    - hardware RAID
    - replication: CephFS/GlusterFS/HDFS
    - Software RAID/erase code: GlusterFS

Lustre storage system HA

- Much effort on high availability
  - req replay mechanism to handle recovery
  - build-in failover mechanism to reconnect to standby servers
- But not so much on high reliability
  - disk array hardware RAID to avoid data loss after disk fails.
  - external disk array + mirror channel
- Unable to tackle
  - Disk array failure
    - The controller failure
    - disk array is unable to be rebuilt
  - Internal disk array
  - No RAID controller Cards

Lustre high reliability design

- HSM
  - backup data to other storage systems
- DAOS snap and clone
  - do napshot and clone container
- Client-driven file-level RAID
  - SNS(Server Network Striping)
  - File level replication



Practices on SNS in lustre-2.4(RAID1)

- Key ideas
  - add raid1 ops in struct **lov_layout_operations**
  - implement RAID1 object operations in every layer of CLIO
- Problems
  - page management across multiple layers
    - VFS page/cl_page/cache
  - Lock management
    - cl_lock stat transfer
    - LDLM lock handle
  - Layout transfer
  - Some reverse operations from bottom to up
    - cl_io_for_each_reverse(scan, io)
    - CL_PAGE_INVOKE()
    - osc_io_submit()->cl_page_prep()->cl_page_invoke()->vvp_page_prep_read()

Lessons learned from our early practice

- The logic is too complex to keep the code stable
  - should handle cl_{io, lock, page, req}
  - should handle LDLM lock
  - should consider about VFS semantics
- Page management is difficult and page cache policy is limited
  - Pages are allocated in VFS but operated in CLIO
  - should fit into OSC page cache policy
- The RAID op error is difficult to be detected
  - async IO
  - page merge in osc request

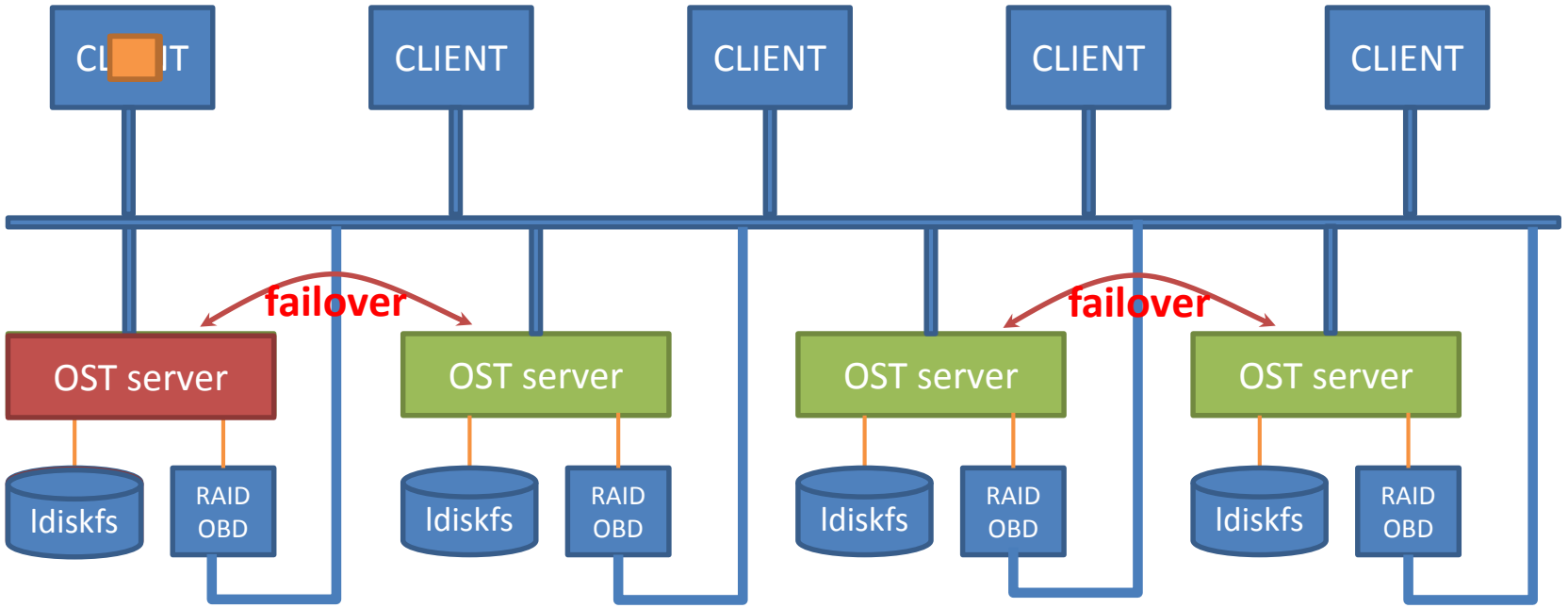
Agenda

- Background
  - motivations
  - practices on client-driven file-level RAID
- Server-driven file-level RAID
  - fundamentals and key ideas
  - overview of the architecture
  - advantages and limitations
- Current state of development
- Future work

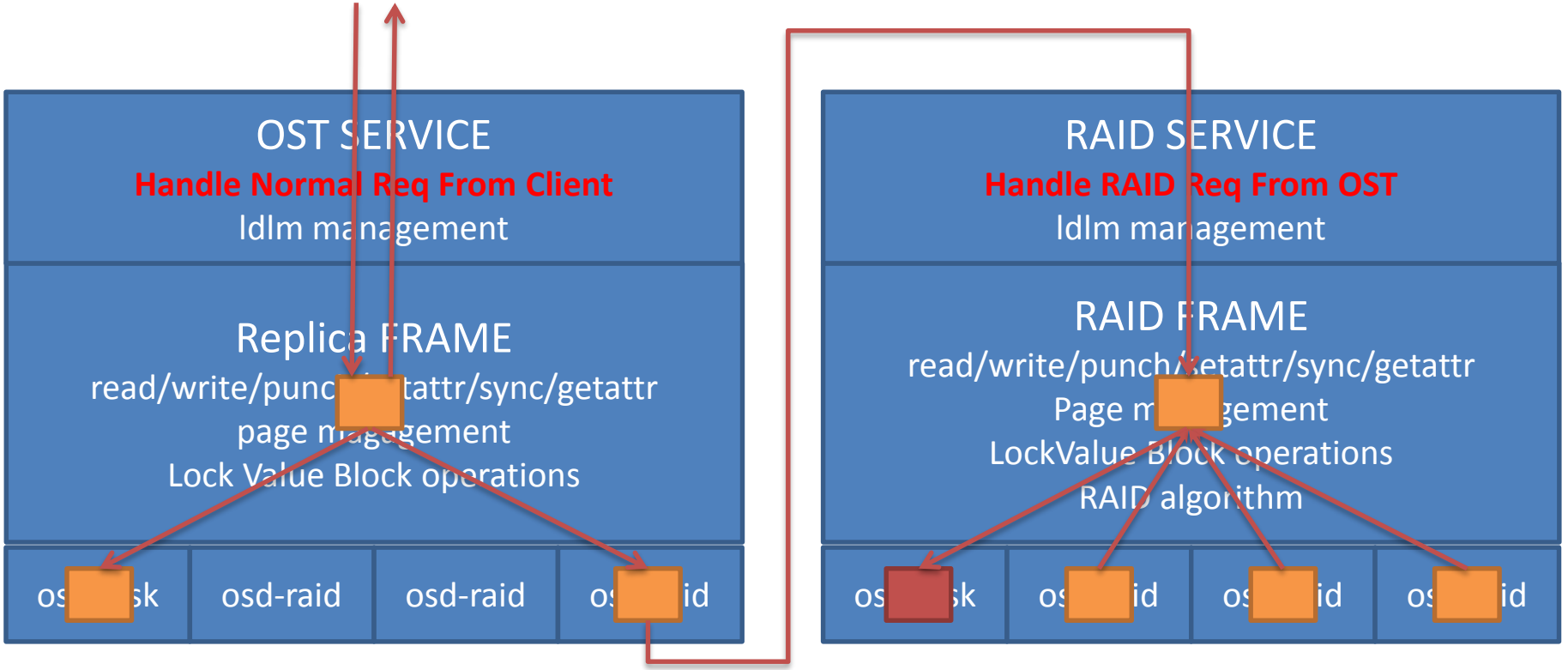
Fundamentals and key ideas

- Fundamentals
  - simple data process logic
  - implemented as a separated and optional module
  - develop a rapid basic prototype of file-level RAID1
  - more software RAID algorithms to support
  - more features to explore
- Key ideas
  - Handle RAID object request on OST servers
  - Leverage the current Lustre HA architecture

Overview of our server-driven file-level RAID architecture



Server-driven file-level RAID stack

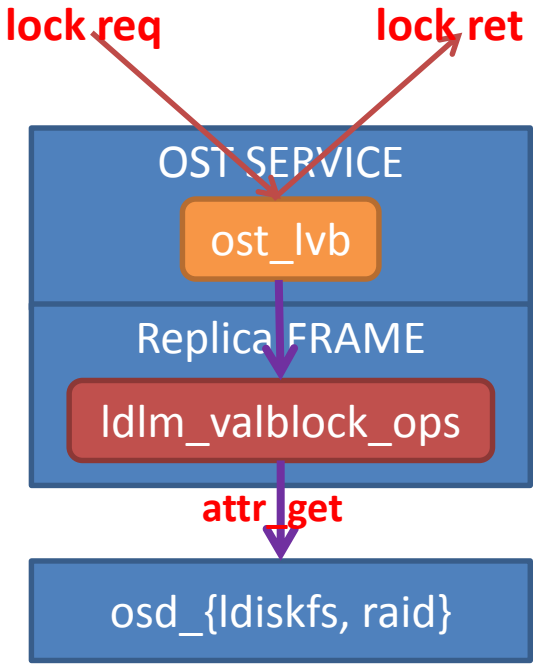


Server-driven file-level raid vs. client-driven file-level RAID

- Advantages
  - LDLM lock operations
  - page management in replica/RAID FRAME
  - Distributed transactions
  - other features to explore
- Limitations

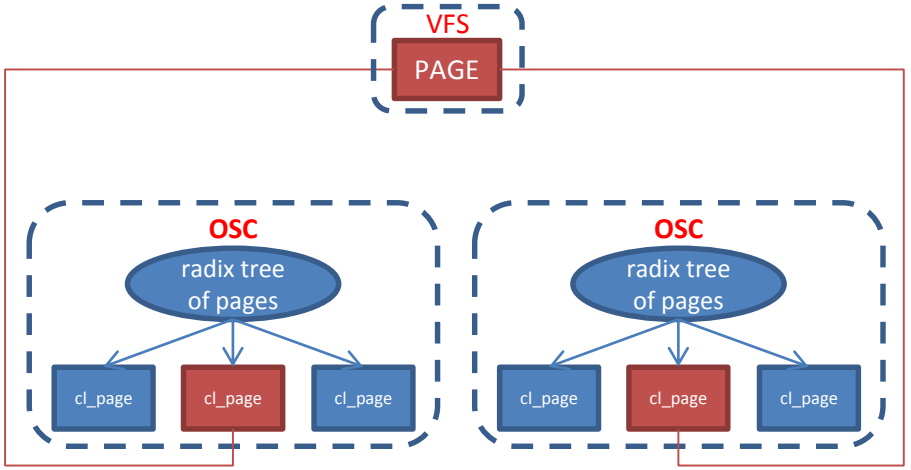
Avoid complicated ldlm lock operations

- Client-driven file-level RAID
  - Different lock type
    - extent lock
    - glimpse lock
    - lockless
  - Complicated lock operations
    - cl_lock ops
    - Lockless ops
    - Ldlm lock callback
    - Lock merge
- Server-driven file-level RAID
  - Ldlm lock is handled by OST service
  - Replica FRAME just need to support LVB operations

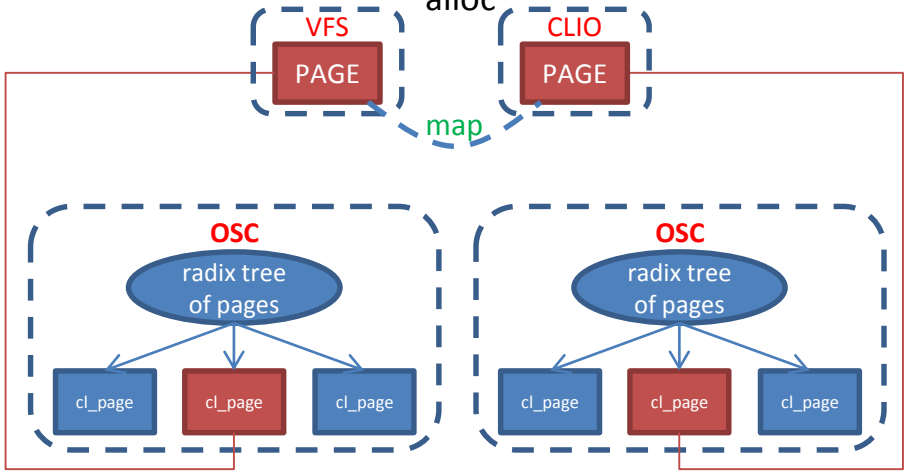


More simple and intelligent page management

- Page management on client
  - should keep VFS semantics
    - page status
    - VFS lock on page
    - file cache in VFS
  - should handle cl_page at every layer
    - {ccc, lov}_page and page ops
    - {lovsub, osc}_page and page ops
  - should handle lock with page
  - page can be cached at OSC
- Client-driven file-level RAID
  - difficult to handle page status on shared page
  - difficult to map the page status to shadow page
  - difficult to support async RAID operations



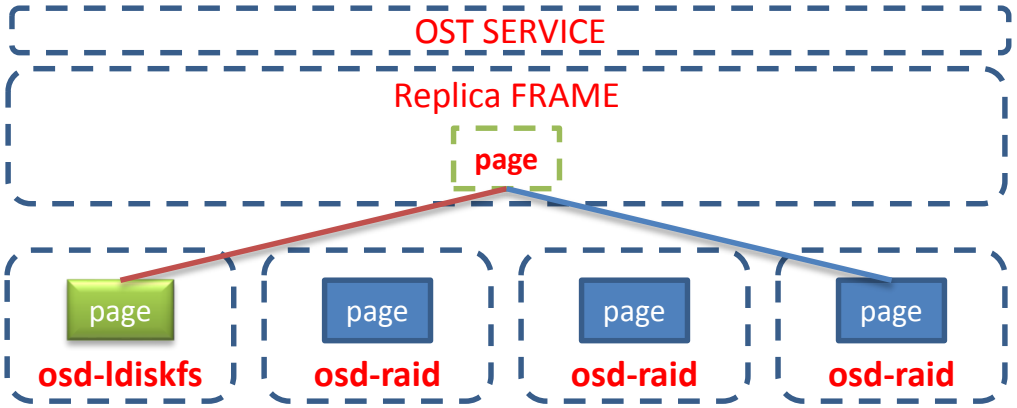
client-driven file-level RAID1 without shadow page alloc



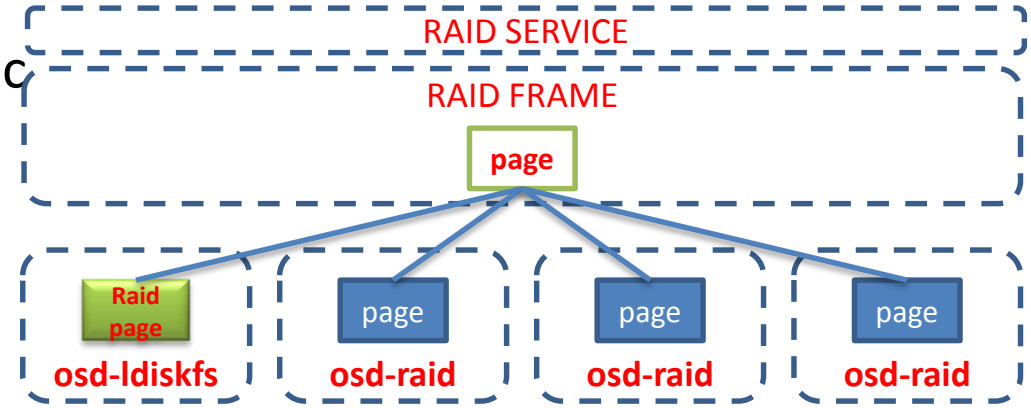
client-driven file-level RAID1 with shadow page alloc

More simple and intelligent page management

- Server-driven file-level RAID
  - pages are used by OST/RAID servcie, no need to keep VFS semantics
  - can easily build mapping among different pages
  - no need to consider LDLM lock
  - be able to implement async RAID operations
  - Be able to add some customized page cache policies



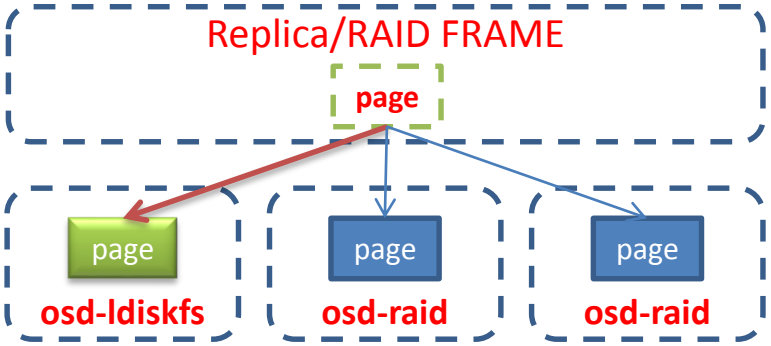
page management IN Replica FRAME



page management in RAID FRAME

Potential to support distributed transaction

- Problem Statement
  - When operation is related to more than one sub-operations on RAID objects (ex. RAID6), we ensure that either both operations succeed or both fail.



- Client-driven file-level RAID
  - no persistent storage
  - difficult to support distributed transaction
- Server-driven file-level RAID
  - have persistent storage
  - leverage transaction mechanism in object storage device API

Other features

- Less code added/modified on Lustre client
  - keep the stability of client code
  - old client can also connect to RAID servers
- RAID algorithm is running on OST server
  - do not consume extra resource (memory, network, CPU) on client
  - take advantages of large memory, high-throughout network and powerful CPU on OST to improve performance

Limitations

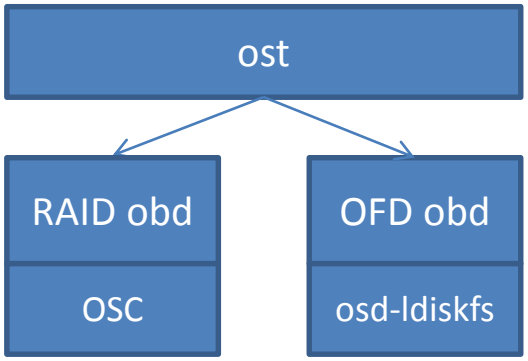
- Network congestion
  - When client and OST on the same node, RAID object ops will race with normal client ops
- To much connections between OSS servers
  - each OSS should connect to the others
- Difficult to improve read performance using raid object
  - local disk read is faster than remote network read
- May get poor performance on other RAID algorithm (ex. RAID5) **???**
  - Doubble data transfer in network
  - RAID object will be updated many times
  - **Performance is as good as it in RAID1**

Agenda

- Background
  - motivations
  - practices on client-driven file-level RAID
- Server-driven file-level RAID
  - fundamentals and key ideas
  - overview of the architecture
  - advantages and limitations
- Current state of development
- Future work

Current state of development

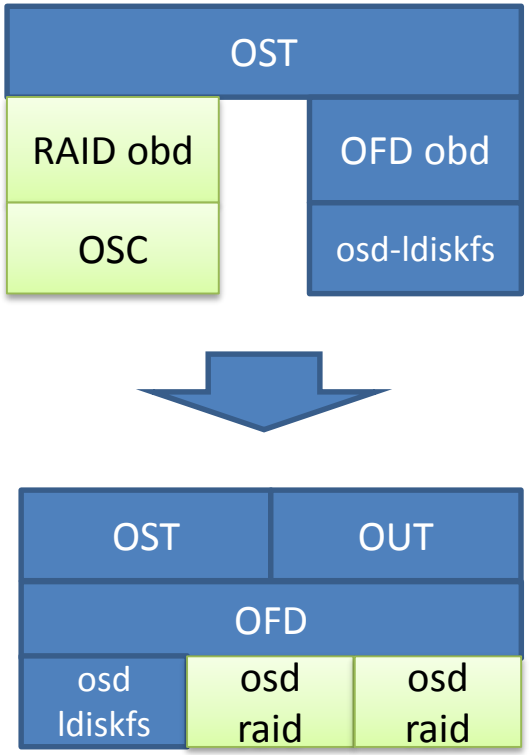
- Development is based on lustre-2.5.2
- Now just focus on file-level RAID1
- The work has been completed
  - file layout support RAID1
  - lfs {set, get}stripe on RAID1 file
  - RAID1 object create and destroy
  - RAID obd build-up
  - page management in raid obd
  - RAID1 object read/write
  - RAID1 object punch/setattr
  - RAID1 lvb ops
  - LDLM lock operations
    - {Glimpes, extent} lock on RAID1 object
    - ost_blocking_ast on RAID1 object
  - File layout lock callback
  - Record stripe status(bad, lost) in RAID1 file layout(struct lov_mds_md)



Now, the system can run smoothly on only raid obd!!!

Future work

- Bad RAID1 file repair and recovery
- Combine with Lustre failover mechanism
- Support quota in RAID1 file
- Reconstruct raid-frame based on osd API
  - use DT API to implement osd-raid device
  - extend the functions of **OSP** device
    - add struct dt_body_operations:do_attr_get
    - add struct dt_object_operations method
  - use **OUT** to handle RAID object ops
- Explore other RAID algorithms



Q & A

Thanks