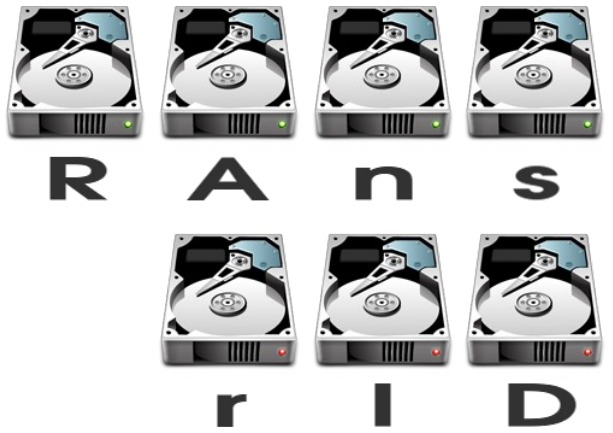
RAnsrID

Redundant Array of Non-Striped Really Independent Disks

Matthias Hopf

SUSE R&D / Novell



SUSE
A NOVELL COMPANY

Background

- Disks like to fail
  - In the worst possible moment (Murphy)
  - With no backup available
  - Restoring backups can take ages
- Use redundancy
  - Store data on more devices than strictly necessary
  - Trade space for security
  - To compensate for one disk loss data on at least one additional disk is required

Failure Possibilities

- Erasures
  - Disk failures, disks unavailable
  - Bad data blocks are known
- Errors
  - Data corruption (MTBF)
  - Bad data blocks are unknown
- Write hole
  - Writes require access to multiple disks
  - Power failure during writes → error
  - Typically stays unnoticed

What is RAnsrID?

- Redundant Array
  - Massively redundant disk array, e.g. 240 data, 16 redundancy disks...
- Non-Striped
  - Data disks still distinguishable
  - Optimized power usage in low use scenarios
- Really Independent Disks
  - Data disks can be mounted solo

What is RAnsrID good for?

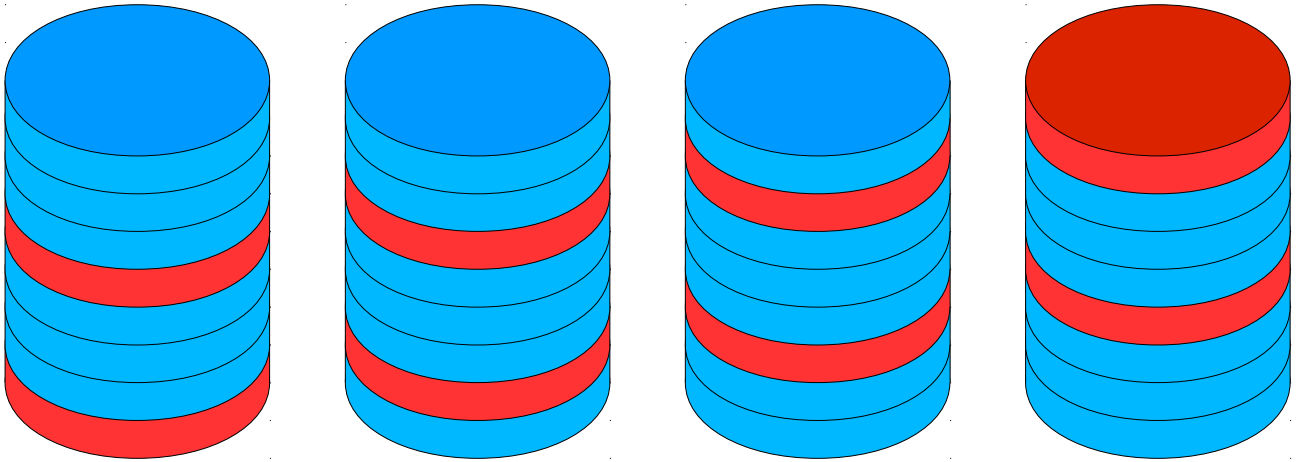
- Low access high volume data
  - Low speed secure disk based backup
  - Data archives
  - Multiple disks spanning media libraries
  - Highly reliable but low access storage
- NOT a general replacement for RAID or even system disks
  - Low speed, user-space block device
  - Suggested to use RAID for journal disk

RAnsrID vs. RAID: Comparison

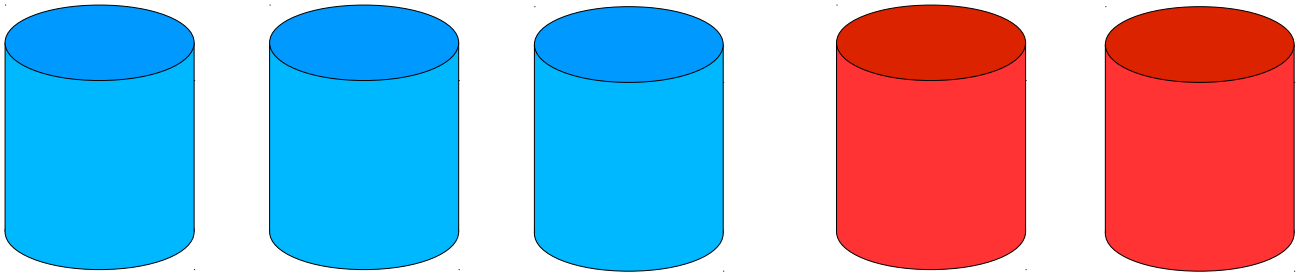
- Basically similar, but:
  - Implemented in user space
  - More redundancy possible
  - Live adding / removing data + redun. disks without array rebuilding
  - Live detaching / attaching disks
  - Only one disk running on data access
  - Possibility to mount data disks solo (r/o)
  - Suited for use with cheap USB disks
  - Higher data integrity, no write hole (future)
  - Much lower speed, higher CPU costs

RAnsrID vs. RAID: Array Layout

RAID 5



RAnsrID



RAnsrID vs. RAID: Redundancies

- For equivalent of m data disks:
  - RAID 0: m disks (No redundancy)
  - RAID 1: 2m disks (Mirroring)
  - RAID 5: m+1 disks:
    XOR of all data
  - RAID 6: m+2 disks:
    XOR + 1 Reed-Solomon, coeffs 2^i
  - RAnsrID: m+n disks:
    n Reed-Solomon blocks,
    coeffs based on Vandermonde matrix

RAnsrID vs. RAID: Error Resilience

- Correctable failures on m disks
  - (RAID 0: None)
  - RAID 1: 1 disk erasure, no errors (m=2)
  - RAID 5: 1 disk erasure, no errors
  - RAID 6: 2 disk erasures or 1 disk error
  - RAnsrID: With n redundancy disks:
    n disk erasures or n/2 disk errors
- Not correctable failures
  - RAID: All data lost
  - RAnsrID: Data on good disks still ok

Basis for Reed-Solomon

- m data **x** written on m+n disks **d**:
  $\mathbf{d} = \mathbf{V} \cdot \mathbf{x}$
- If all m•m submatrices are rank m
  → any m values of **d** are enough to reconstruct **x** (invertible)
- The Vandermonde matrix $\mathbf{V}_{ij} = i^{j-1}$ has this property
- Create unity submatrix in first m rows by subtracting cols (col-based Gauss Jordan)
  → Data explicitly stays unaltered on disks
- Do all that in Galois Field GF(2^8)

Finite Field / Galois Field

- Different field than standard rational numbers
- Limited amount of numbers, e.g. $GF(2^8)$ has 0-255
- Add *and* subtract numbers by XOR: 6 – 5 = 6 + 5 = %110 xor %101 = 3
- Multiplication more difficult (modulated ring shift); use tables
- Constructed by polynomial multiplication and division

Reed-Solomon in RAnsrID (1)

- Read: Fast
  - Data available due to unity submatrix
- Write: Slow
  - Read old data value and redundancies
  - Remove contribution to reds.
  - Add contribution of new data to reds.
  - Write out new data and redundancies
- Adding new redundancy disks
  - Read all data values
  - Calculate redundancies and write

Reed-Solomon in RAnsrID (2)

- Reconstruction
  - Invert submatrix of available disks
  - Read available disks
  - Calculate unavailable/erroneous disks
  - Write disks
  - Optionally validate
- Validation
  - Read all disks
  - Calculate redundancies from data
  - Compare

Disk Layout

- RAnsrID metadata in last block of device
- n copies of the block at begin of device
  - Otherwise disk will be auto-detected by most Desktops and mounted → writes to disk
  - Writes directly to disk invalidate redundancy disks
  - When solo mount with **mount -oro,offset=4096** …

Metadata Layout

- Data required for RAnsrID
  - Magic, version, CRC, name
  - Vandermonde parameters
  - UIDs for diskset and disk
  - Disk number (Vandermonde matrix line)
  - Size, offset of data
  - Logical time of last change
  - Disk condition
  - Coeffs for redundancy disks

Disk states

- Disk condition
  - good: All fine
  - bad/broken: Disk has bad/broken blocks
  - zero: New disk, all zero'ed
  - nonmember: Not part of array
- Disk status
  - on: active
  - off: not active, system can activate it
  - n/a: not available (temp. removed)
  - fail: configuration failure

Block Tracking

- Bad vs. broken
  - bad: known erasure
  - broken: earsure/error, requires user interaction
- Failed blocks are tracked
  - May change e.g. during writes on n/a disks
  - Multiple lists to track bad blocks
  - May concatenate blocks with small gaps

Validation and Reconstruction

- Validation
  - Only on user request
  - Currently no detection which disks are corrupted (only the error itself)
  - Erasures marked automatically
- Reconstruction
  - Currently only on user request
  - Later automatic for bad block ranges
  - Semantics of bad vs. broken needs more thought

Journal

- Data integrity
  - Journal disk state changes, detected bad blocks
  - Journal writes to avoid write hole (future)
  - Journal adding/removing disks (future)
- Startup speedup
  - Detect partial rebuilds, enable restart where last left
- NOT related to file system journals

Network Block Device

- Implemented as NBD server
  - User space server, connects via TCP with client kernel
  - Visible on client kernel as block device
  - Multiple disks visible as GPT partitions on this device
  - Note: server and client running on same machine used to freeze on writes
  - Fixed with 2.6.26
  - Only one write enabled client active
  - Addon protocol to control server

Usage: Administrative

- Create RAnsrID disks:
  ransrid_admin c [dev] [opts]
  - Required: **–u** [uid]
    Array unique identifier
  - Required: **–d** [nr]
    Disk number >= 0 data disk
    < 0 redundancy disk
  - Optional: **–0**
    Clear disk for fast including in array
- Destructive – double check device!
  - Does not check for mounts yet

Usage: Administrative (2)

- Modify RAnsrID disks:
  ransrid_admin m [dev] [opts]
  - Change condition, Reed-Solomon coeffs, uids, etc.
- Careful! You get what you ask for..
  - Suggested to use online commands to server instead
  - Also helps keeping the journal in sync
  - Easy to shoot oneself in the virtual foot

Usage: Administrative (3)

- Create RAnsrID journal:
  ransrid_admin J -S 4
  /var/tmp/ransrid.journal
  - Minimum journal size: 4MB
  - Currently not much reason to use larger
  - Will change with write journaling
- View journal anytime:
  ransrid_admin j
  /var/tmp/ransrid.journal

Usage: Server

- Start server:
  ransrid_server [devs]
  - Currently no live adding of devices
  - Devices and journal need to be created first
  - Journal currently always /var/tmp/ransrid.journal
  - Listens on port 2000, standard nbd protocol
  - Asks for confirmation if meta data changed, acts r/o if denied

Usage: Server (2)

```
Validating journal: [...]
Devices:
Disk -1    ^-1 uid 0x9e89d0c3  [ok] [ON]  10240MB
           tm 1025   /dev/sdb1
           Redundancy for 0-1
Disk 0     ^0 uid 0x8f560fde   [ok] [ON]  10240MB
           tm 1026   /dev/sdc1
Disk 1     ^1 uid 0x8cae73e5   [ok] [ON]  10240MB
           tm 1027   /dev/sdd1

Array UID:            0x00001234
Array data disks:  2 total  2 ok  2 active
Array red disks:   1 total  1 ok  1 active[...]
Starting nbd server...
```

Usage: NBD Client + Mounting

- Load nbd kernel module with partition support:
  modprobe nbd nbds_max=4 max_part=16
- Start NBD client:
  nbd-client bs=512 [server] **2000 /dev/nbd0**
- Mount partitions:
  mount /dev/nbd0p1 /media/1

Usage: Controlling the Server

- All done with test client:
  ransrid_test [server] **2000 [cmd]**
- **l**
  Show server status continuously
- **c** [dev] [cond]
  Set condition of device nr [dev]:
  “ok” “bad” “broken”
- **m** [dev] [offset] [num] [type]
  Set condititon for block ranges only

Usage: Controlling the Server (2)

- **V 0 0**
  Validate whole array.
  - Alternatively specify block range
- **R 0 0**
  Repair all bad/broken blocks
  - Alternative specify block range
- **S** [name] [val]
  Set server behavior variables

Server Behavior

- Intended to control run time behavior and automatic handling of errors
- Currently only one working

  r/o Server is read only if set

Practice

So let's head for the demo...

Novell

TODO

- Live disk enable/disablement
- Live reconstruction
- Hardware failure handling
- Erroneous disk detection
- Rigorous testing
- Write + add/remove journaling
- Optimization
- Cleanup + documentation
- Much more...

Q + A

Questions?

Answers!

Code on
git@gitorious.org:ransrid/ransrid.git

Patches welcome :-)