



ZFS

The Last Word in Filesystem

chwong

What is RAID?



RAID

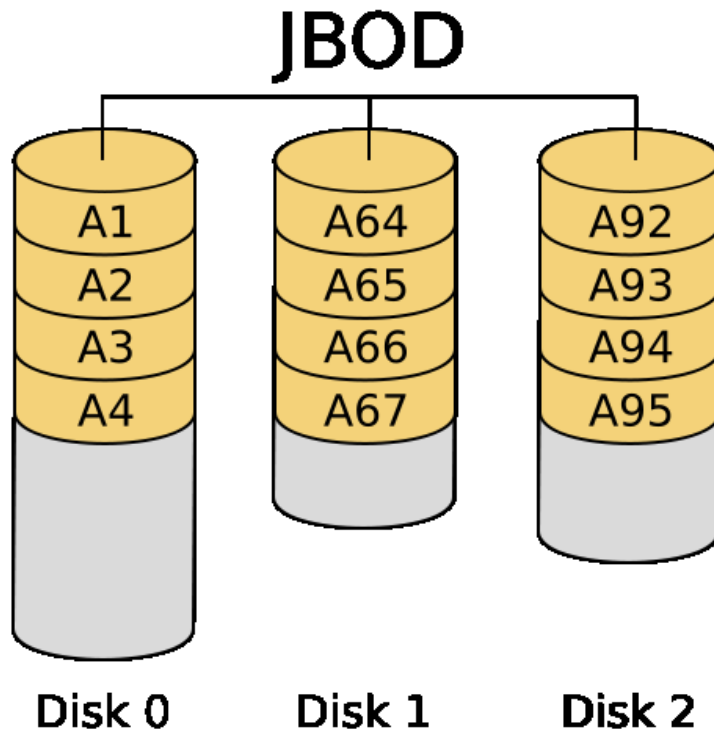
- ❑ Redundant Array of Independent Disks
- ❑ A group of drives glue into one



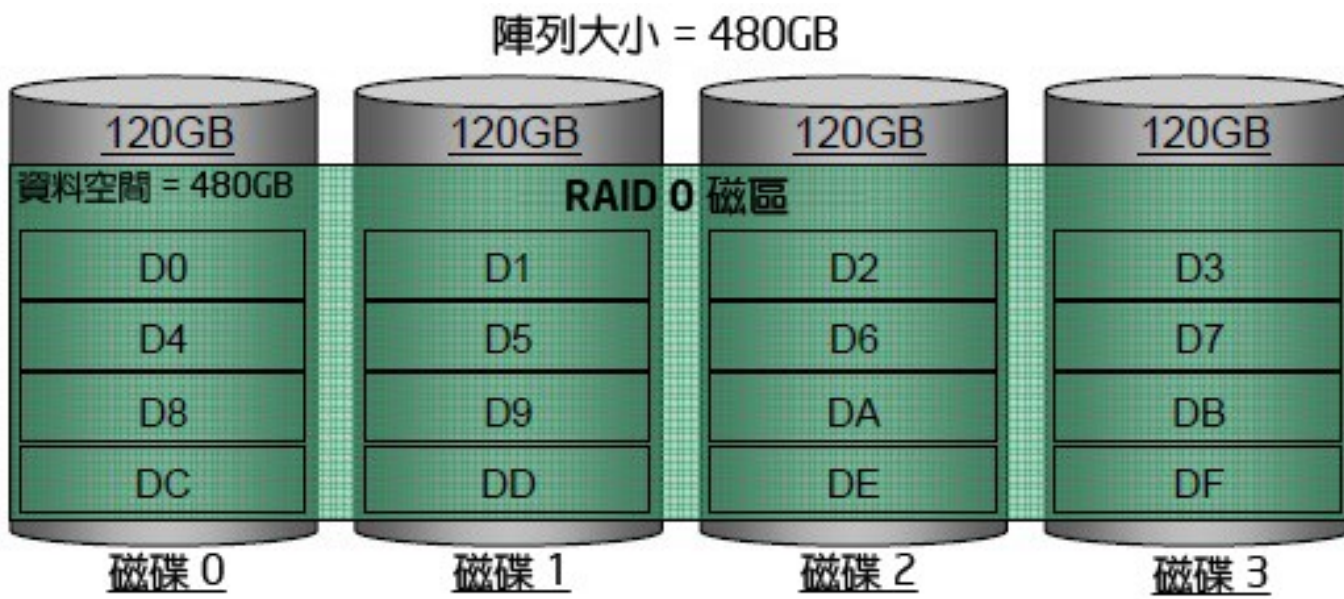
Common RAID types

- JBOD
- RAID 0
- RAID 1
- RAID 5
- RAID 6
- RAID 10?
- RAID 50?
- RAID 60?

JBOD (Just a Bunch Of Disks)



RAID 0 (Stripe)

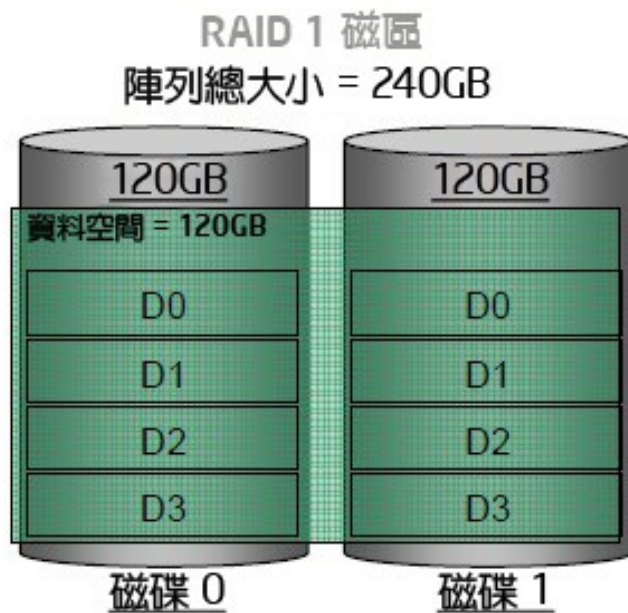


RAID 0 (Stripe)

- Striping data onto multiple devices
- 2X Write/Read Speed

- Data corrupt if ANY of the device fail

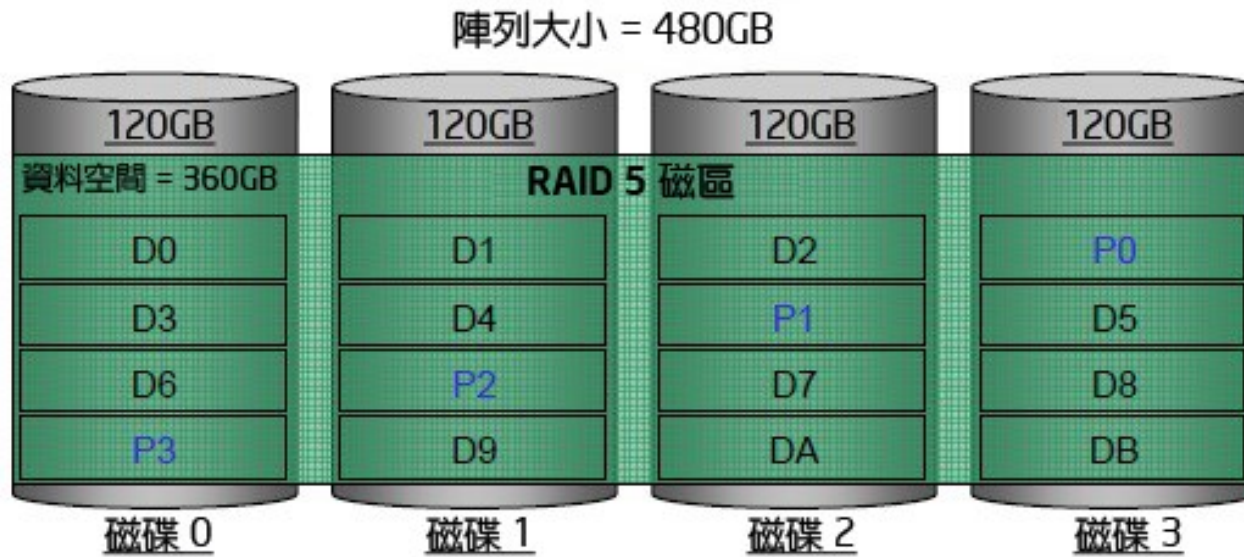
RAID 1 (Mirror)



RAID 1 (Mirror)

- Devices contain identical data
- 100% redundancy
- Fast read

RAID 5

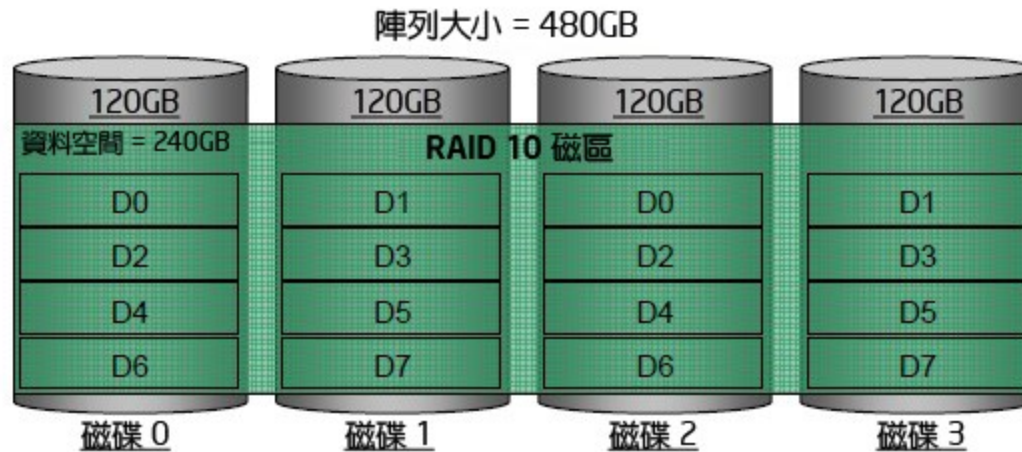


RAID 5

- Slower the raid 0 / raid 1
- Higher cpu usage

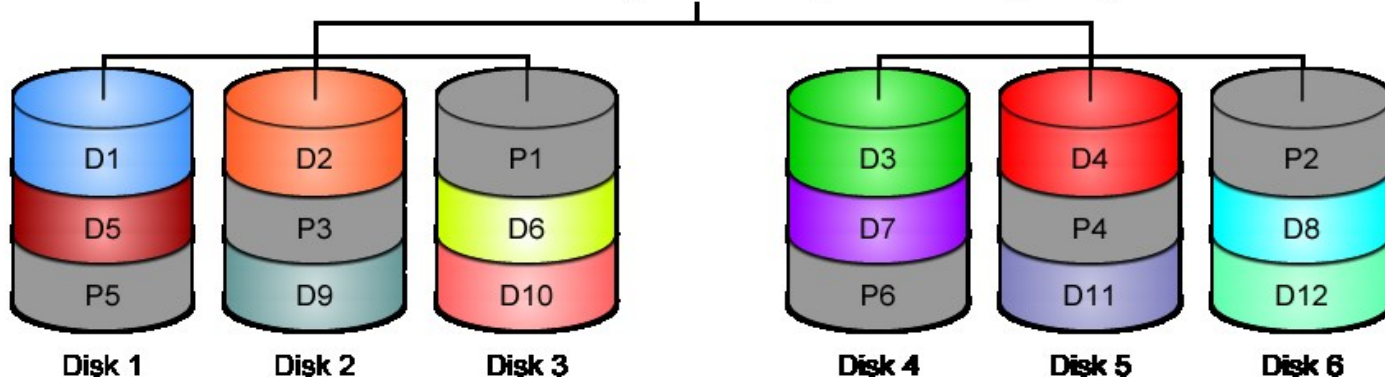
RAID 10?

❑ RAID 1+0



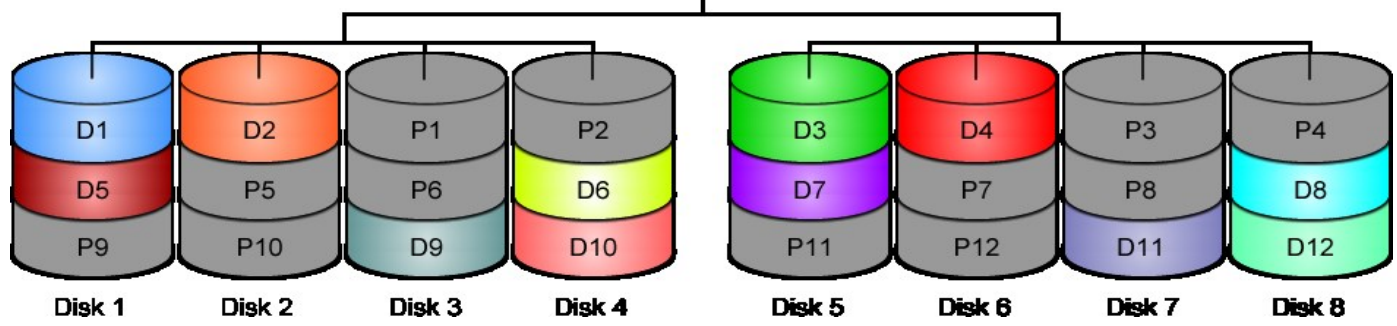
RAID 50?

RAID 50 (Parity+Stripe)



RAID 60?

RAID 60 (Double Parity+Stripe)





Here comes ZFS

Why ZFS?

- ❑ Easy administration
- ❑ Highly scalable (128 bit)
- ❑ Transactional Copy-on-Write
- ❑ Fully checksummed
- ❑ Revolutionary and modern
- ❑ SSD and Memory friendly

ZFS Pools

- ❑ ZFS is not just filesystem
- ❑ ZFS = filesystem + volume manager

- ❑ Work out of the box
- ❑ Zuper zimple to create
- ❑ Controlled with single command
 - zpool

ZFS Pools Components

- ❑ Pool is create from vdevs (Virtual Devices)
- ❑ What is vdevs?
 - ❑ **disk**: A real disk (sda)
 - ❑ **file**: A file
 - ❑ **mirror**: Two or more disks mirrored together
 - ❑ **raidz1/2**: Three or more disks in RAID5/6*
 - ❑ **spare**: A spare drive
 - ❑ **log**: A write log device (ZIL SLOG; typically SSD)
 - ❑ **cache**: A read cache device (L2ARC; typically SSD)

RAID in ZFS

- ❑ ***Dynamic Stripe***: Intelligent RAID0
- ❑ **Mirror**: RAID 1
- ❑ **Raidz1**: Improved from RAID5 (parity)
- ❑ **Raidz2**: Improved from RAID6 (double parity)
- ❑ **Raidz3**: triple parity
- ❑ Combined as dynamic stripe

Create a simple zpool

❑ `zpool create mypool /dev/sda /dev/sdb`

```
Dynamic Stripe (RAID 0)
  |- /dev/sda
  |- /dev/sdb
```

❑ `zpool create mypool`

- `mirror /dev/sda /dev/sdb`
- `mirror /dev/sdc /dev/sdd`

❑ What is this?

WT* is this

```
zpool create mypool
  mirror /dev/sda /dev/sdb
  mirror /dev/sdc /dev/sdd
    raidz /dev/sde /dev/sdf /dev/sdg
      log mirror /dev/sdh /dev/sdi
        cache /dev/sdj /dev/sdk
          spare /dev/sdl /dev/sdm
```

Zpool command

zpool list

list all the zpool

zpool status [pool name]

show status of zpool

zpool export/import [pool name]

export or import given pool

zpool set/get <properties/all>

set or show zpool properties

zpool online/offline <pool name> <vdev>

set an device in zpool to online/offline state

zpool attach/detach <pool name> <device> <new device>

attach a new device to an zpool/detach a device from zpool

zpool replace <pool name> <old device> <new device>

replace old device with new device

zpool scrub

try to discover silent error or hardware

failure

zpool history [pool name]

show all the history of zpool

zpool add <pool name> <vdev>

add additional capacity into pool

zpool create/destroy

create/destory zpool

Zpool properties

Each pool has customizable properties

NAME	PROPERTY	VALUE	SOURCE
zroot	size	460G	-
zroot	capacity	4%	-
zroot	altroot	-	default
zroot	health	ONLINE	-
zroot	guid	13063928643765267585	default
zroot	version	-	default
zroot	bootfs	zroot/ROOT/default	local
zroot	delegation	on	default
zroot	autoreplace	off	default
zroot	cachefile	-	default
zroot	failmode	wait	default
zroot	listsnapshots	off	default

Zpool Sizing

- ❑ ZFS reserve 1/64 of pool capacity for safe-guard to protect CoW
- ❑ RAIDZ1 Space = Total Drive Capacity -1 Drive
- ❑ RAIDZ2 Space = Total Drive Capacity -2 Drives
- ❑ RAIDZ3 Space = Total Drive Capacity -3 Drives
- ❑ Dynamic Stripe of 4* 100GB= 400 / 1.016= ~390GB
- ❑ RAIDZ1 of 4* 100GB = 300GB - 1/64th= ~295GB
- ❑ RAIDZ2 of 4* 100GB = 200GB - 1/64th= ~195GB
- ❑ RAIDZ2 of 10* 100GB = 800GB - 1/64th= ~780GB
- ❑ <http://cuddletech.com/blog/pivot/entry.php?id=1013>



ZFS Dataset

ZFS Datasets

- ❑ Two forms:
 - filesystem: just like traditional filesystem
 - volume: block device

- ❑ Nested
- ❑ Each dataset has associated properties that can be inherited by sub-filesystems
- ❑ Controlled with single command
 - zfs

Filesystem Datasets

- ❑ Create new dataset with
 - `zfs create <pool name>/<dataset name>`

- ❑ New dataset inherits properties of parent dataset

Volume Datasets (ZVols)

- ❑ Block storage
- ❑ Located at `/dev/zvol/<pool name>/<dataset>`
- ❑ Used for iSCSI and other non-zfs local filesystem
- ❑ Support “thin provisioning”

Dataset properties

NAME	PROPERTY	VALUE	SOURCE
zroot	type	filesystem	-
zroot	creation	Mon Jul 21 23:13 2014	-
zroot	used	22.6G	-
zroot	available	423G	-
zroot	referenced	144K	-
zroot	compressratio	1.07x	-
zroot	mounted	no	-
zroot	quota	none	default
zroot	reservation	none	default
zroot	recordsize	128K	default
zroot	mountpoint	none	local
zroot	sharenfs	off	default

zfs command

zfs set/get <prop. / all> <dataset>

set properties of datasets

zfs promote

promote clone to the origin of filesystem

zfs create <dataset>

create new dataset

zfs send/receive

send/receive data stream of snapshot
with pipe

zfs destroy

destroy datasets/snapshots/clones..

zfs snapshot

create snapshots

zfs rollback

rollback to given snapshot

Snapshot

- ❑ Natural benefit of ZFS's Copy-On-Write design
- ❑ Create a point-in-time “copy” of a dataset
- ❑ Used for file recovery or full dataset rollback
- ❑ Denoted by @ symbol

Create snapshot

❑ # zfs snapshot tank/something@2015-01-02

- done in seconds
- no additional disk space consume

Rollback

- ❑ `# zfs rollback zroot/something@2015-01-02`
 - IRREVERSIBLY revert dataset to previous state
 - All more current snapshot will be destroyed

Recover single file?

- hidden “.zfs” directory in dataset mount point
- set snapdir to visible

Clone

- ❑ “copy” a separate dataset from a snapshot
- ❑ caveat! still dependent on source snapshot

Promotion

- ❑ Reverse parent/child relationship of cloned dataset and referenced snapshot
- ❑ So that the referenced snapshot can be destroyed or reverted

Replication

❑ # zfs send tank/somethin@123 | zfs recv

- dataset can be piped over network
- dataset can also be received from pipe



Performance Tuning

General tuning tips

- System memory
- Access time
- Dataset compression
- Deduplication
- ZFS send and receive

Random Access Memory

- ❑ ZFS performance depends on the amount of system
 - recommended minimum: 1GB
 - 4GB is ok
 - 8GB and more is good

Dataset compression

- Save space
- Increase cpu usage
- Increase data throughput

Deduplication

- ❑ requires even more memory
- ❑ increases cpu usage

ZFS send/recv

- ❑ using buffer for large streams
 - misc/buffer
 - misc/mbuffer (network capable)

Database tuning

- ❑ For PostgreSQL and MySQL users recommend using a different recordsize than default 128k.
- ❑ PostgreSQL: 8k
- ❑ MySQL MyISAM storage: 8k
- ❑ MySQL InnoDB storage: 16k

File Servers

- Disable access time
- keep number of snapshots low
- dedup only if you have lots of RAM
- for heavy write workloads move ZIL to separate SSD drives
- optionally disable ZIL for datasets (beware consequences)

Webservers

❑ Disable redundant data caching

- Apache
 - EnableMMAP Off
 - EnableSendfile Off
- Nginx
 - Sendfile off
- Lighttpd
 - `server.network-backend="writev"`



Cache and Prefetch

ARC

Adaptive Replacement Cache

Resides in system RAM

major speedup to ZFS

the size is auto-tuned

Default:

arc max: memory size - 1GB

metadata limit: $\frac{1}{4}$ of arc_max

arc min: $\frac{1}{2}$ of arc_meta_limit (but at least 16MB)

Tuning ARC

- ❑ Disable ARC on per-dataset level
- ❑ maximum can be limited
- ❑ increasing `arc_meta_limit` may help if working with many files

- ❑ `# sysctl kstat.zfs.misc.arcstats.size`
- ❑ `# sysctl vfs.zfs.arc_meta_used`
- ❑ `# sysctl vfs.zfs.arc_meta_limit`

- ❑ <http://www.krausam.de/?p=70>

L2ARC

❑ L2 Adaptive Replacement Cache

- is designed to run on fast block devices (SSD)
- helps primarily read-intensive workloads
- each device can be attached to only one ZFS pool

❑ # zpool add <pool name> cache <vdevs>

❑ # zpool add remove <pool name> <vdevs>

Tuning L2ARC

enable prefetch for streaming or serving of large files

configurable on per-dataset basis

turbo warmup phase may require tuning (e.g. set to 16MB)

`vfs.zfs.l2arc_noprefetch`

`vfs.zfs.l2arc_write_max`

`vfs.zfs.l2arc_write_boost`

ZIL

❑ ZFS Intent Log

- guarantees data consistency on fsync() calls
- replays transaction in case of a panic or power failure
- use small storage space on each pool by default

❑ To speed up writes, deploy zil on a separate log device(SSD)

❑ Per-dataset synchronicity behavior can be configured

- # zfs set sync=[standard|always|disabled] dataset

File-level Prefetch (zfetch)

- ❑ Analyses read patterns of files
- ❑ Tries to predict next reads

- ❑ Loader tunable to enable/disable zfetch:
`vfs.zfs.prefetch_disable`

Device-level Prefetch (vdev prefetch)

- ❑ reads data after small reads from pool devices
- ❑ useful for drives with higher latency
- ❑ consumes constant RAM per vdev
- ❑ is disabled by default

- ❑ Loader tunable to enable/disable vdev prefetch:
`vfs.zfs.vdev.cache.size=[bytes]`

ZFS Statistics Tools

```
# sysctl vfs.zfs
```

```
# sysctl kstat.zfs
```

using tools:

zfs-stats: analyzes settings and counters since boot

zfsf-mon: real-time statistics with averages

Both tools are available in ports under sysutils/zfs-stats

References

❑ ZFS tuning in FreeBSD (Martin Matuška):

- Slide
 - <http://blog.vx.sk/uploads/conferences/EuroBSDcon2012/zfs-tuning-handout.pdf>
- Video
 - <https://www.youtube.com/watch?v=PIpI7Ub6yjo>

❑ Becoming a ZFS Ninja (Ben Rockwood):

- <http://www.cuddletech.com/blog/pivot/entry.php?id=1075>

❑ ZFS Administration:

- <https://pthree.org/2012/12/14/zfs-administration-part-ix-copy-on-write>