

Tuning ZFS on FreeBSD

Martin Matuška

`mm@FreeBSD.org`

EuroBSDCon 2012
21.10.2012



ZFS is a modern 128-bit open-source operating system utilizing the copy-on-write model.

This presentation is going to cover the following topics:

- ▶ How can we tune ZFS?
- ▶ When should ZFS be tuned?

Is ZFS slow?

Help, my ZFS is slow!

- ▶ compared to ... ?
- ▶ this depends on many factors (workload, data access, ...)
- ▶ tradeoff speed vs data consistency + features
- ▶ maybe auto-tuning does not well in your case ...

Think twice about what you do

From blogs about optimizing ZFS:

Disable the unwanted features

By default, ZFS enables a lot of settings for data security, such as checksum etc. If you don't care about the additional data security, just disable them.

<http://icesquare.com/wordpress/how-to-improve-zfs-performance>

A note on disabling ZFS checksum: don't.

<http://v-reality.info/2010/06/using-nexentastor-zfs-storage-appliance-with-vsphere>

ZFS checksum

We need the checksums to do:

- ▶ data and metadata integrity verification
- ▶ self-healing (scrub, mirror, raidz)

General tuning tips

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

Random Access Memory

ZFS performance and stability depends on the amount of system RAM.

- ▶ recommended minimum: 1GB
- ▶ 4GB is ok
- ▶ 8GB and more is good

Access time

Due to copy-on-write enabled access time:

- ▶ reduces performance on read-intensive workloads
- ▶ increases space used by snapshots

```
# zfs set atime=off dataset
```

Dataset compression

Dataset compression does:

- ▶ save space (LZJB less, gzip more)
- ▶ increase CPU usage (LZJB less, gzip much more)
- ▶ increase data throughput (LZJB, relative)

Therefore I recommend using compression primarily for archiving purposes (e.g. system or webserver logfiles)

```
# zfs set compression=[on|off|gzip]
```

Deduplication

Deduplication saves space by keeping just a single copy of identical data blocks. But remember, that deduplication:

- ▶ requires even more memory (fast only if table fits to RAM)
- ▶ increases CPU usage

Command to simulate the effect of enabling deduplication:

```
# zdb -S pool
```

Command for viewing detailed deduplication information:

```
# zdb -D pool  
# zdb -DD pool
```

ZFS send and receive

I highly recommend using an intermediate buffering solution for sending and receiving large streams:

- ▶ misc/buffer
- ▶ misc/mbuffer (network capable)

Application Tuning Tips

We are going to look how to optimize the following applications for ZFS:

- ▶ Web servers
- ▶ Database servers
- ▶ File servers

Webservers

As of the current ZFS implementation it is recommended to disable sendfile and mmap to avoid redundant data caching.

- ▶ Apache

```
EnableMMAP Off
```

```
EnableSendfile Off
```

- ▶ Nginx

```
Sendfile off
```

- ▶ Lighttpd

```
server.network-backend="writev"
```

Database servers

For PostgreSQL and MySQL users recommend using a different recordsize than default 128k.

- ▶ PostgreSQL: 8k
- ▶ MySQL MyISAM storage: 8k
- ▶ MySQL InnoDB storage: 16k

```
# zfs create -o recordsize=8k tank/mysql
```

File Servers

Here are some tips for 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)

Cache and Prefetch Tuning

We are going to look at the following:

- ▶ Adaptive Replacement Cache (ARC)
- ▶ Level 2 Adaptive Replacement Cache (L2ARC)
- ▶ ZFS Intent Log (ZIL)
- ▶ File-level Prefetch (zfetch)
- ▶ Device-level Prefetch (vdev prefetch)
- ▶ ZFS Statistics Tools

Adaptive Replacement Cache 1/2

ARC resides in system RAM, provides major speedup to ZFS and its size is auto-tuned.

Default values are:

- ▶ maximum: physical RAM less 1GB (or 1/2 of all memory)
- ▶ metadata limit: `arc_meta_limit = 1/4 of arc_max`
- ▶ minimum: 1/2 of `arc_meta_limit` (but at least 16MB)

Adaptive Replacement Cache 2/2

How to tune the ARC:

- ▶ you can disable ARC on per-dataset level
- ▶ maximum can be limited to reserve memory for other tasks
- ▶ 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
```

Level 2 Adaptive Replacement Cache 1/2

Some facts about L2ARC:

- ▶ 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 cache device
# zpool remove pool device
```

Level 2 Adaptive Replacement Cache 2/2

How to tune the 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
```

ZFS Intent Log

The ZFS Intent Log (ZIL)

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

To speed up writes, you can deploy ZIL on a separate log device.
Per-dataset synchronicity behaviour can be configured.

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

File-level Prefetch (zfetcch)

File-level prefetching

- ▶ analyses read patterns of files
- ▶ tries to predict next reads
- ▶ goal: reduce application response times

Loader tunable to enable/disable zfetcch:

`vfs.zfs.prefetch_disable`

Device-level Prefetch (vdev prefetch)

Device-level prefetching

- ▶ reads data after small reads from pool devices
- ▶ may be 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

ZFS statistical data is provided by

```
# sysctl vfs.zfs  
# sysctl kstat.zfs
```

This data can help to make tuning decisions.

I have prepared tools to view and analyze this data:

- ▶ **zfs-stats**: analyzes settings and counters since boot
- ▶ **zfs-mon**: real-time statistics with averages

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

`zfs-stats`: overview

The `zfs-stats` utility is based on Ben Rockwood's `arc-summary.pl` and includes modifications by Jason J. Hellenthal and myself.

It provides information about:

- ▶ ARC structure and efficiency
- ▶ L2ARC structure and efficiency
- ▶ ZFETCH efficiency
- ▶ values of ZFS tunables
- ▶ system memory (overview)

`zfs-stats`: sample output excerpt

ARC Size:	79.89%	25.57	GiB
Target Size: (Adaptive)	79.89%	25.57	GiB
Min Size (Hard Limit):	12.50%	4.00	GiB
Max Size (High Water):	8:1	32.00	GiB
ARC Efficiency:		1.25b	
Cache Hit Ratio:	90.52%	1.13b	
Cache Miss Ratio:	9.48%	118.08m	
Actual Hit Ratio:	84.54%	1.05b	
Data Demand Efficiency:	95.45%	356.90m	
Data Prefetch Efficiency:	40.64%	11.36m	
L2 ARC Breakdown:		118.18m	
Hit Ratio:	62.87%	74.29m	
Miss Ratio:	37.13%	43.89m	
Feeds:		849.64k	
File-Level Prefetch: (HEALTHY)			
DMU Efficiency:		28.09b	
Hit Ratio:	88.54%	24.87b	

zfs-mon: overview

The zfs-mon utility

- ▶ polls ZFS counters in real-time
- ▶ analyzes ARC, L2ARC, ZFETCH and vdev prefetch
- ▶ displays absolute and relative values
- ▶ displays output in varnishstat(1) style

ZFS real-time cache activity monitor
Seconds elapsed: 120

Cache hits and misses:

	1s	10s	60s	tot
ARC hits:	259	431	418	466
ARC misses:	51	40	49	52
ARC demand data hits:	223	417	390	437
ARC demand data misses:	36	20	17	16
ARC demand metadata hits:	36	11	25	25
ARC demand metadata misses:	15	19	21	25
ARC prefetch data hits:	0	4	3	4
ARC prefetch data misses:	0	1	10	8
ARC prefetch metadata hits:	0	0	0	0
ARC prefetch metadata misses:	0	0	1	3
L2ARC hits:	47	34	40	37
L2ARC misses:	4	5	9	15
ZFETCH hits:	47903	47294	48155	47138
ZFETCH misses:	272	449	1147	3593

Cache efficiency percentage:

	10s	60s	tot
ARC:	91.51	89.51	89.96
ARC demand data:	95.42	95.82	96.47
ARC demand metadata:	36.67	54.35	50.00
ARC prefetch data:	80.00	23.08	33.33
ARC prefetch metadata:	0.00	0.00	0.00
L2ARC:	87.18	81.63	71.15
ZFETCH:	99.06	97.67	92.92

Thank you for your attention!