ZFS 分层架构设计

Table of Contents

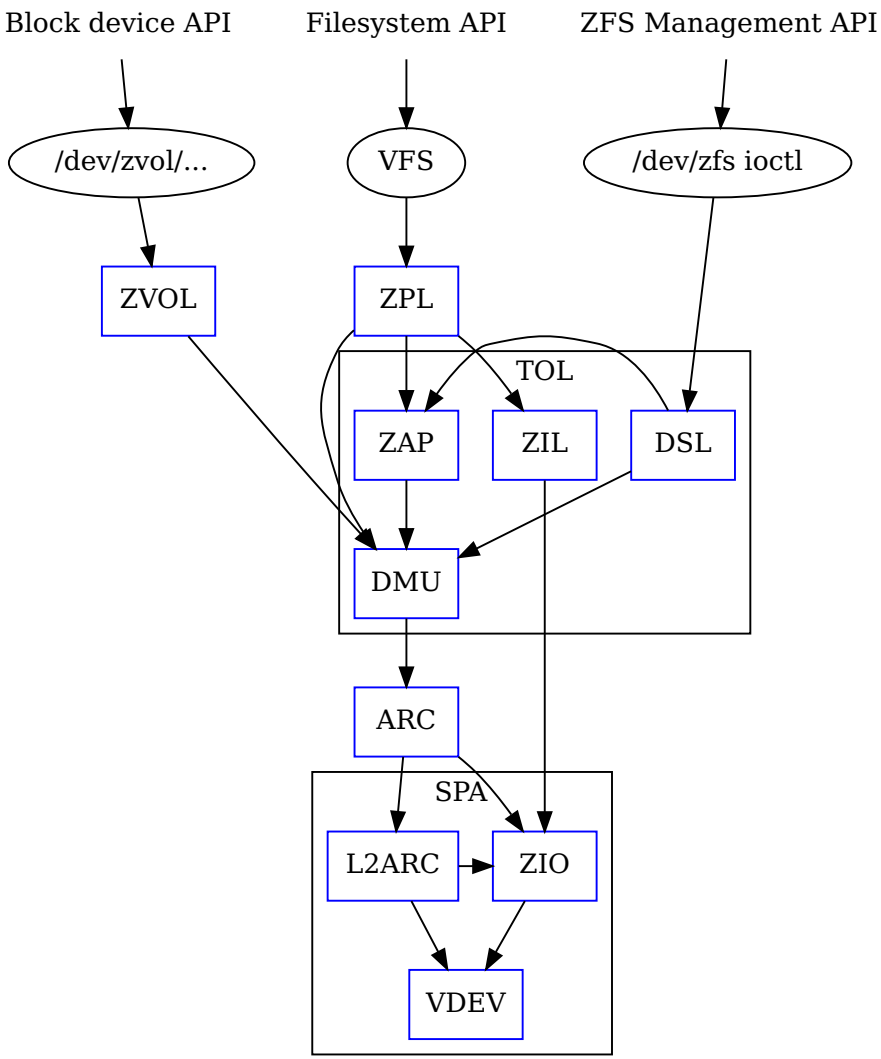
Contents

- ZAP
- ZIL
- DSL
- DMU
- ARC
- SPA

ZFS 在设计之初背负了重构 Solaris 诸多内核子系统的重任，从而不同于 Linux 的文件系统 只负责文件系统的功能而把其余功能（比如内存脏页管理，IO调度）交给内核更底层的子系统， ZFS 的整体设计更层次化并更独立，很多部分可能和 Linux 内核已有的子系统有功能重叠。 而本文想讲的只是 ZFS 中与快照相关的一些部分，于是先从 ZFS 的整体设计上说一下和快照相关的概念位于 ZFS 设计中的什么位置。

子系统整体架构

首先 ZFS 整体架构如下图，其中圆圈是 ZFS 给内核层的外部接口，方框是 ZFS 内部子系统：


Block device API
Filesystem API
ZFS Management API
/dev/zvol/...
VFS
/dev/zfs ioctl
ZVOL
ZPL
TOL
ZAP
ZIL
DSL
DMU
ARC
SPA
L2ARC
ZIO
VDEV

VDEV

Virtual DEVice

作用相当于 Linux Device Mapper 层或者 FreeBSD GEOM 层，提供 Stripe/Mirror/RAIDZ 之类的多设备存储池管理和抽象。 ZFS 中的 vdev 形成一个树状结构，在树的底层是从内核提供的物理设备， 其上是虚拟的块设备。每个虚拟块设备对上对下都是块设备接口。

ZIO

ZFS I/O，作用相当于内核的 IO scheduler 。

ZPL

ZFS Posix Layer，提供符合 POSIX 文件系统的语义，也就是包括文件、目录这些抽象以及 inode 属性、权限那些，对一个普通 FS 而言用户直接接触的部分。

ZVOL

ZFS VOLume

有点像 loopback block device，暴露一个块设备的接口，其上可以创建别的 FS。对 ZFS 而言实现 ZVOL 的意义在于它是比文件更简单的接口所以一开始先实现的它，而且 早期 Solaris 没有 sparse 文件的时候可以用它模拟很大的块设备，测试 Solaris UFS 对 TB 级存储的支持情况。

TOL

Transactional Object Layer

在数据块的基础上提供一个事务性的对象语义层。每个对象用多个数据块存储，每个数据块大概是 4K~128K 这样的数量级。

ZAP

ZFS Attribute Processor ，在「对象」基础上提供紧凑的 name/value 映射，从而文件夹内容、文件属性之类的都是基于 ZAP 。

ZIL

ZFS Intent Log ，记录两次完整事务语义提交之间的 log ，用来加速实现 fsync 之类的保证。

DSL

Dataset and Snapshot Layer ，数据集和快照层，这是本文的重点。

DMU

Data Management Unit，在块的基础上提供「对象」的抽象。每个「对象」可以是一个文件，或者是别的 ZFS 内部需要记录的东西。

ARC

Adaptive Replacement Cache，作用相当于 pagecache 。

SPA

Storage Pool Allocator，从内核的多个块设备中抽象出存储池。