文件系统的宏观结构

前两篇笔记分别记录了 硬盘 和 闪存 两大类存储设备的物理特性。 这些存储设备，在操作系统中被抽象为块设备（block device）， 是因为他们不同于随机存储设备（Random Access Memory），需要按块为单位读写。 文件系统构建在块设备抽象之上，需要根据存储设备的物理特性做设计和优化，满足性能和安全性两方面的考量。

对传统硬盘而言，寻址时间依赖于地址跳转间隔，地址跳转越远寻址时消耗的时间越长。 因此设计文件系统时要尽量避免远距离的地址跳转，让读写尽量连续。对 SSD 而言，虽然没有了跳转时间的考量，但是随机寻址的写入仍然会加重 FTL 地址翻译的负担， 并且增加垃圾回收时写放大的程度，所以对 SSD 也是同样，按地址连续写入时的性能远优于随机写入。 现代硬盘上内外圈的读写性能也会影响到文件系统设计，外圈柱面的连续读写能比内圈柱面上快很多倍， 甚至一些硬盘的外圈连续读取速度能高于 SSD 的读取速度， 所以在硬盘上写入数据时将经常读写的数据（比如文件系统元数据）放入低端地址能有效提升性能。

现代文件系统通过合理安排存储设备上的数据结构布局，来为这些读写情况优化性能，这种优化叫地理布局优化（ geometry based optimization）。 本篇笔记想讨论一下文件系统在存储设备上安排数据时如何做地理布局优化。文件系统设计中这种优化分两方面： 安排数据布局的「宏观结构（macro structures）」和实际分配地址的「分配器（allocators）」算法。

「宏观结构（macro structures）」是盘上数据结构（on-disk structures）中， 关于如何分割并分配整个地址空间的那些数据结构。「分配器（allocators）」则是在文件系统代码中负责 在宏观结构里分配地址安放数据的具体算法。数据结构和算法相互配合而最优化性能， 所以考察文件系统性能时也应当从这两者的设计开始考虑。

值得区别的是「宏观结构」是特定文件系统记录的方式，所以和文件系统代码的实现无关， 不同操作系统下实现同一个文件系统要支持同样的宏观结构布局，也难以在保持兼容性的前提下修改宏观结构。 而「分配器算法」则属于具体文件系统实现的代码，可能在使用同样盘上结构的情况下优化算法、改善分配器性能。 不同操作系统下实现的同一个文件系统也可以采用不同的算法。

DOS文件系统与文件分配表（FAT12 FAT16 VFAT FAT32）

FAT 系文件系统最初为软盘驱动器设计，

早期 Unix 文件系统

柱面-磁头-扇区寻址与地理位置优化

BSD 的 FFS

Linux 的 ext2

物理寻址到逻辑块寻址

Flash 媒介与 FTL