Part 1 学习日记

最近在学习操作系统的实践过程中学习了一些有关文件存储的相关知识，在此稍作记录。

最近在安装linux操作系统时学习了一些记录一下学习的文件的存储相关知识

储存介质

特点

- 磁带
  - 储存密度大，成本低
  - 读写速度很慢
  - 只能顺序读写
  - 可覆盖写
  - 目前主要用于存储归档资料（？）
- 较为传统的机械磁盘
  - 储存密度较大，成本较低
  - 读写速度较慢
  - 可随机读写，寻址成本较高
  - 可覆盖写
- SSD（指由主控芯片、高速缓存和闪存芯片构成的高速固态硬盘）
  - 储存密度较低，成本较高（不断发展中……）
  - 读写快
  - 可随机读写，寻址成本低
  - 硬件特性：不可覆盖写
- SMR 机械硬盘
  - 相较于传统的机械硬盘，可实现更高的储存密度和更低的成本
  - 硬件特性：不可随机写

细节

- 机械硬盘
  - 柱面、磁头、扇区
  - 按逻辑上连续的扇区顺序读写速度最快（但这些扇区在物理层面上不一定是连续的）
- SSD
  - 通过主控芯片和缓存来隐藏不可覆盖写的特性（地址重映射）
  - 由于只能整块擦出，存在写入放大的问题
  - 好的固件可以实现磨损均衡，冷、热数据分离等特性
- SMR
  - 由于写入时会影响到相邻的磁道，只能顺序地写入空区域
  - 需要较大的缓存以便对数据进行整理
  - 极大量的随机写入时表现极为糟糕

    想想一边要读写盘上现有的块，一边还要处理来自操作系统的新的写操作……画面不要太美（不过日常使用中好像也很少遇到需要大量连续的随机写的情景）

文件系统

- 功能
  - 磁盘空间管理
    - 总空间
    - 可用空间统计
    - 可用空间分布
  - 文件信息记录
    - 文件内容
    - 其他数据
      - 路径
      - 文件名
      - 时间戳
      - 权限相关信息
  - 文件系统的结构与文件索引信息
- 数据结构：B-Tree
  - 特点：在每个节点储存较多信息（至少填满一个块），可以有效降低树的层数，从而减少磁盘寻址次数，提高文件系统性能
  - 衍生的数据结构：
    - B+-Tree
      - 增加了同一层元素之间的指针，遍历效率更高
    - B*-Tree
      - 通过一定条件下节点的合并将最差情况下的空间利用率由1/2提高到2/3

数据安全

文件系统面临着随时掉电的风险，这要求一个可靠的文件系统必须能在随时掉电的情况下维持自身的一致性。为了维持在特殊情况下的文件安全，专家们不同的思路。

- 文件日志

  文件日志是一种可以一定程度上保障数据安全的方法。为了防止意外掉电所造成的损失，在真正将所需要写入磁盘的内容写入正确的位置之前，会先把这次操作记录在“日志区”，在等待记录完成之后才真正去执行写入的操作。这样，在掉电之后就可以根据日志区的内容恢复之前未完成的操作了。

  文件日志的问题在于会增加写入操作，降低磁盘的写入性能。此外，写入日志的过程中、利用日志恢复的过程中如果遭遇突然掉电，仅凭日志也不能完全保障数据的安全。

  一般使用的日志记录方式仅记录元数据日志，在性能与安全之中取得了较好的平衡。

- COW (Copy-On-Write)

  COW 是一个新颖、不同的文件系统实现方式。它不仅可以在数据安全方面有所作为，而且利用COW也可以在文件系统层面实现许多有趣的特性，如快照、合并重复数据等等。

- 日志结构文件系统

  前一段时间频频被提到的 F2FS 便是日志结构文件系统的一种。它把储存介质视为只能追加写的设备，由此将所有的写入操作合并为连续写入，大大提高了随机写入的速度。但储存空间毕竟有限，完全写入一次之后，为了保证仍有大量的连续空间可供使用，便需要对之前写过的空间进项垃圾回收处理。这一垃圾处理的过程是这类文件系统的技术难点。不够优秀的垃圾回收方法可能会大大降低文件系统性能。

Part 2 实践操作

笔者最近学习并使用linux操作系统的过程中，由于对系统的结构和功能不够熟悉，对个方面状况也不够了解，因而时常出现由于错误操作、错误配置而出现系统部分功能损坏甚至整个系统无法启动的状况。笔者希望能够拥有可靠的备份，以便在出现错误之后方便、快速地回复到正常状态。

由于这些错误操作几乎全部发生于根目录下，而 COW 可以方便地实现整个文件系统级别的快照，遂将根目录转换为 btrfs 文件系统。

btrfs 是基于 COW 的文件系统，其快照作为独立的子卷可以只读或读写状态单独挂载。通过将不同时期产生的快照挂载到根目录，即可实现系统的回滚。

在实际的操作过程中，注意到一下几点：

1. 将根目录与其快照分别放置于不同子卷当中可更加方便管理，应避免将快照子卷置于根目录之下，易产生混乱。
2. 产生快照时应产生只读快照，在需要回滚时再对只读快照创建可写快照并将其挂载为根目录，防止再次错误后可用快照被修改，陷入无可用快照的困境。
3. 由于被快照引用的文件被删除后不会释放空间，被修改后会增加空间占用，因此在磁盘空间不充足时应留意剩余空间，适量使用快照。此外，btrfs 的快照文件组织形式使得快照（子卷）的删除是一个成本极高的操作，需要较长的时间才能完全完成。因此，btrfs 工具在删除子卷后会立即提供反馈，同时标注(no-commit)，示意该操作未提交到磁盘。

btrfs 还具有可在线改变体积（增大&缩小）、透明压缩等特性，在使用的过程之中极为灵活。

基于如此强大、丰富的特性，笔者试图实现类似 Mac OS 中时光机的功能：定期将操作系统的所有数据备份在仓库盘上。

快照功能固然强大、快速，但快照与在线的文件系统引用同一份文件，这意味着若在磁盘层面发生数据损坏，快照并不能对我们有所帮助。因此，备份还是必要的。

实现备份的基本思路便是创建如下定时任务：创建当前文件系统快照，并将快照内容发送至仓库盘进行备份。但每次都传输全盘文件的成本过高，增量备份更是一个可行的方案。在搜索过后，btrbk 很好地利用了 btrfs 丰富的特性 (send and receive) 实现了笔者的需求。

EOF