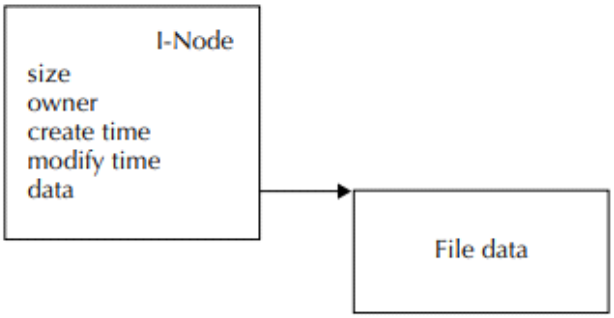
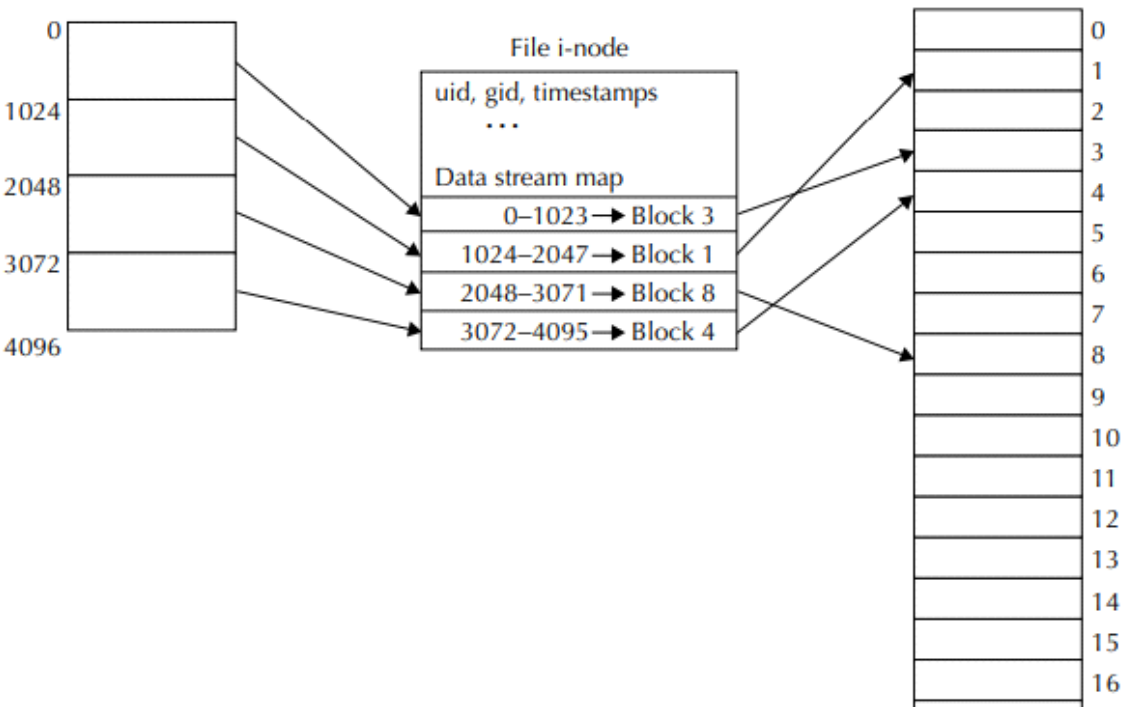
文件系统设计参考

2018年5月17日　　11:01

1. 文件系统是什么？
文件系统主要的目的就是在存储介质上创建，修改，读取和存储数据。更高一层的抽象，文件系统就是来组织管理和读取存储物理介质上的信息的。主要用来简化永华使用接口，屏蔽底层硬件差异。
2. 文件系统中的一些术语和基础概念
- Disk：物理介质抽象
- Block：最小存取单元
- Partition/namespace：a subset of all the blocks on a disk
- Volume:a set of logic blocks on a disk or crossed disks
- Superblock:磁盘块信息管理，一般是个bitmap
- Metadata：管理数据
- Data：用户数据
- Journal/log：数据修改操作日志
- Inode：文件或者目录用来管理数据块的一个结构抽象（文件名，时间戳，访问控制和归属信息，数据块信息）



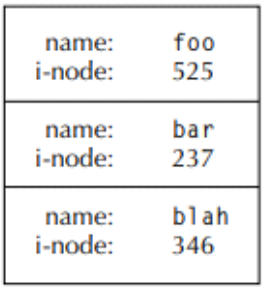
- Blockmap：数据块映射管理结构



- Extent：数据块映射管理结构
  大文件比较友好，多个连续logic block由一个extent信息管理与文件流的映射。
- Attribute：文件属性信息

3. 文件系统抽象
   - File 字节流
     a. name -> data 映射（可以使用KV管理）
     b. Metadata （参见章节2 inode）
     c. Data （参见章节2 blockmap&extent）

   - Directory
     a. File list



     b. Directory tree
        扁平结构和树形结构：链表，B-tree，hash table，主要考虑查找效率和插入删除效率。

4. 文件系统的一些基础操作
   - 初始化
     mkfs初始化一个空的文件系统，指定blocksize，通过bitmap管理可用和已用的block空间，分区信息记录，journal信息记录。将所有信息持久化到superblock中。创建root目录并持久化到superblock中。
   - Mount
     读取superblock信息，检查磁盘status，构建内存缓存并执行一致性状态校验，日志回放。
   - Umount
     一般是做flush操作，保证所有内存缓存和dirty数据刷盘。superblock标记正常卸载。
   - Create file
     创建name到inode的dentry，给inode分配block（是否预分配）
   - Create directry
     和创件文件类似，inode的block内容是file list或者index tree
   - Open file
     第一步是在目录树中做查找操作，如果找到返回对应的inode，通过inode访问数据块。（忽略权限控制）
   - Read file
     <fd, offset, length>
   - Write file
     随机写：
        覆盖：读-修改-写
        不覆盖：在空的block写，或者新申请block写，更新metadata
     append写：申请新的block，修改metadata，写入新的block
   - Delete file
     第一步从目录树中移除，并发控制逻辑，然后将元数据和数据块标记为删除，没有文件引用再进行真实删除，回收block到空闲block区，更新元数据。
   - Rename file
     并发控制逻辑
   - Open directory
     ls功能
   - Read directory
     ls功能
   - 扩展操作（软硬链接，属性，索引和日志，访问控制）

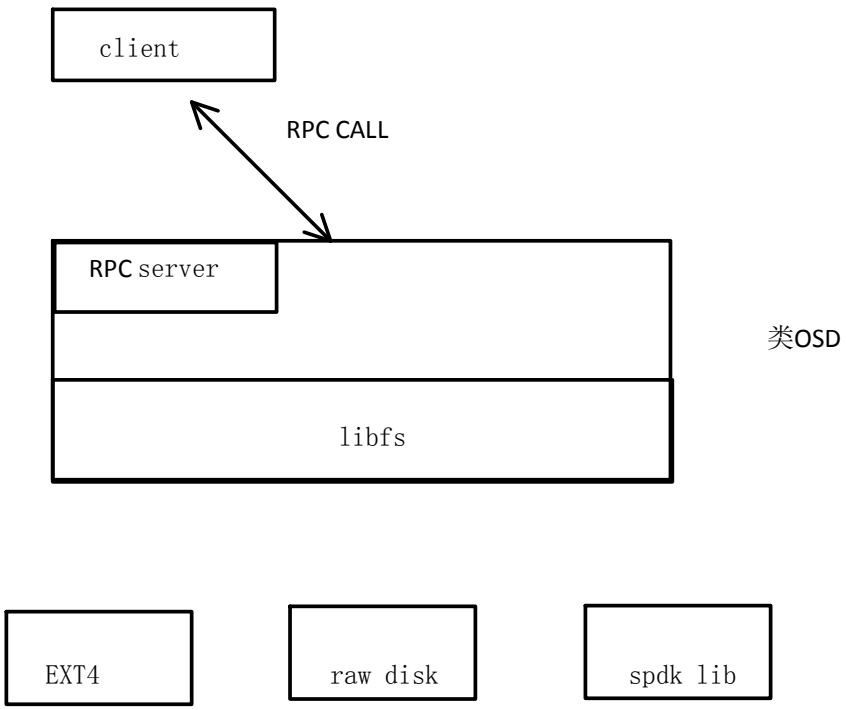
5. 文件系统调研
   - Ext4
   - Bluestore
   - Xfs
   - Lfs

6. 自己开发文件系统的难点和优势在哪？

- 难点
  a. 磁盘空间管理
  b. 元数据管理　　可以借助数据库
  c. 事务支持（日志型文件系统，基于特定写接口的的事务）
  d. 稳定性和可靠性（主要考虑开发难度和工作量，由于比较底层，一般的文件系统开发周期都比较长）
- 优势
  a. 需求相对简单特化
  不需要支持所有的posxi接口，针对特性场景和特性文件的实现。实现逻辑比通用系统比简单，但是基础的管理逻辑都是需要的。
  b. 针对硬件特性优化
  Nvme ssd + spdk，IO性能是有事

7. 基础需求
- 类posix接口，open、close、create、delete、read、write
- 事务支持
- snapshot支持
- 随机写or追加写？
- 目录操作
- debug和运维

8. 技术选型参考
- Bluestore
  元数据交给数据库管理和持久化，事务支持也交给数据库，这两块是文件系统的难点，交给现成的组件降低开发难度。
- Log structured file system
  https://people.eecs.berkeley.edu/~brewer/cs262/LFS.pdf 未分析完
- Rocksdbenv
  底层存储和fs分层，底层可以支持标准文件系统，rawdisk和spdklib，fs提供够用的api接口



9. 文件系统衡量标准
- 元数据大小和占用磁盘比例
- 并发性能
- 读写放大比例

10. 总结
没有最优的文件系统，只有最适合的；文件系统的设计要根据上层的需求和workload来：
1. 读写的频率
2. 顺序访问还是随机访问
3. 文件大小

4. 文件的生命周期等