Linux Performance

文件系统

日志文件系统是怎样工作的

2016/11/10 | VMUNIX

文件系统要解决的一个关键问题是怎样防止掉电或系统崩溃造成数据损坏，在此类意外事件中，导致文件系统损坏的根本原因在于写文件不是原子操作，因为写文件涉及的不仅仅是用户数据，还涉及元数据(metadata)包括 Superblock、inode bitmap、inode、data block bitmap等，所以写操作无法一步完成，如果其中任何一个步骤被打断，就会造成数据的不一致或损坏。举一个简化的例子，我们对一个文件进行写操作，要涉及以下步骤：

1. 从data block bitmap中分配一个数据块；
2. 在inode中添加指向数据块的指针；
3. 把用户数据写入数据块。

- 如果步骤2完成了，3未完成，结果是数据损坏，因为该文件认为数据块是自己的，但里面的数据其实是垃圾；
- 如果步骤2完成了，1未完成，结果是元数据不一致，因为该文件已经把数据块据为己有，然而文件系统却还认为该数据块未分配、随后又可能会把该数据块分配给别的文件、造成数据覆盖；
- 如果步骤1完成了、2未完成，结果就是文件系统分配了一个数据块，但是没有任何文件用到这个数据块，造成空间浪费；
- 如果步骤3完成了，2未完成，结果就是用户数据写入了硬盘数据块中，但白写了，因为文件不知道这个数据块是自己的。

日志文件系统(Journal File System)就是为解决上述问题而诞生的。它的原理是在进行写操作之前，把即将进行的各个步骤（称为transaction）事先记录下来，保存在文件系统上单独开辟的一块空间上，这就是所谓的日志(journal)，也被称为write-ahead logging，日志保存成功之后才进行真正的写操作、把文件系统的元数据和用户数据写进硬盘（称为checkpoint），这样万一写操作的过程中掉电，下次挂载文件系统之前把保存好的日志重新执行一遍就行了（术语叫做replay），避免了前述的数据损坏场景。

有人问如果保存日志的过程中掉电怎么办？最初始的想法是把一条日志的数据一次性写入硬盘，相当于一个原子操作，然而这并不可行，因为硬盘通常以512字节为单位进行操作，日志数据一超过512字节就不可能一次性写入了。所以实际上是这么做的：给每一条日志设置一个结束符，只有在日志写入成功之后才写结束符，如果一条日志没有对应的结束符就会被视为无效日志，直接丢弃，这样就保证了日志里的数据是完整的。

一条日志在它对应的写操作完成之后就没用了，占用的硬盘空间就可以释放。保存日志的硬盘空间大小是有限的，被循环使用，所以日志也被称为circular log。

至此可以总结一下日志文件系统的工作步骤了：

1. Journal write : 把transaction写入日志中；
2. Journal commit : 在一条日志保存好之后，写入结束符；
3. Checkpoint : 进行真正的写操作，把元数据(metadata)和用户数据(user data)写入文件系统；
4. Free : 回收日志占用的硬盘空间。

以上方式把用户数据(user data)也记录在日志中，称为**Data Journaling**，Linux EXT3文件系统就支持这种方式，这种方式存在效率问题：就是每一个写操作涉及的元数据(metadata)和用户数据(user data)实际上都要在硬盘上写两次，一次写在日志里，一次写在文件系统上。元数据倒也罢了，用户数据通常比较大，拷贝几个GB的电影文件也要乘以2实在是降低了效率。

一个更高效的方式是**Metadata Journaling**，不把用户数据(user data)记录在日志中，它防止数据损坏的方法是先写入用户数据(user data)、再写日志，即在上述"Journal write"之前先写用户数据，这样就保证了只要日志是有效的，那么它对应的用户数据也是有效的，一旦发生掉电故障，最坏的结果也就是最后一条日志没记完，那么对应的用户数据也会丢，效果与Data Journaling丢弃日志一样，重要的是文件系统的一致性和完整性是有保证的。Metadata Journaling又叫Ordered Journaling，大多数文件系统都采用这种方式。像Linux EXT3文件系统也是可以选择Data Journaling还是Ordered Journaling的。

参考资料：
Crash Consistency: FSCK and Journaling