[1]

Extending the Lifetime of Flash-based Storage through Reducing WriteAmplification from File Systems

减少文件系统的写放大→延长闪存存储器的寿命

对于闪存寿命，常用解决方法：磨损均衡/数据减少

但由于

1. 数据减少不应该损害文件系统机制(如在数据缩减中不应删除日志中的重复数据)
2. 为保持一致性大多数元数据频繁同步→阻止数据的减少

**则 文件系统中的元数据更新很难减少**

→上述数据减少技术对减少文件系统的写放大无效

**文件系统的常用机制加剧了闪存写放大问题**

1. 日志：用于保持更新的原子性

因 数据/元数据会复制到日志中→使写大小加倍

1. 元数据写入同步：为避免数据损失

元数据占空间极小但频繁写入造成巨大的写入流量→加剧内存磨损

1. 页面对齐更新(对小更新亦是)

局部页面更新：读取→修改→写入(对于整个页面)

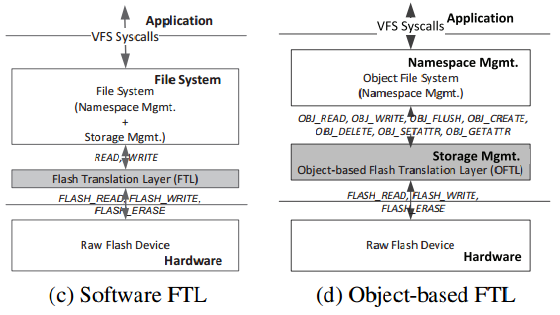
增大页面大小的趋势→加剧该问题

**则 尝试通过减少文件系统的写放大→延长闪存存储器的寿命**

**本文贡献**

1. **基于对象的闪存转换层(OFTL)**→更好的协作文件系统和闪存

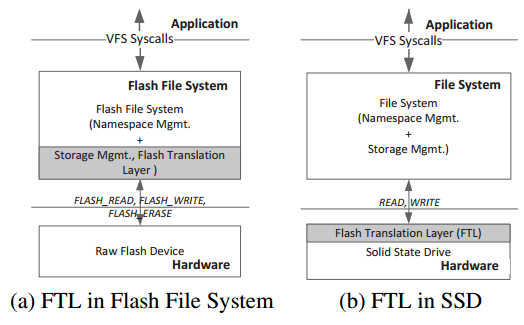
①OFTL的结构优点



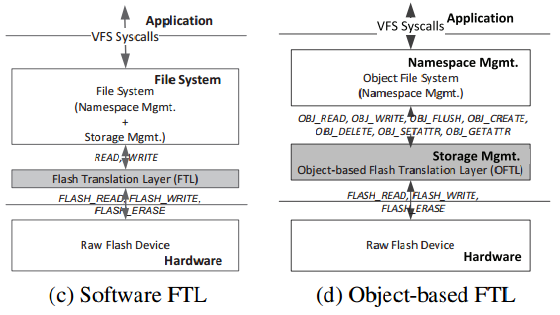
1. OFTL管理存储→直接管理闪存→闪存特性可被用于优化文件系统

/更好的协作文件系统和FTL

1. OFTL管理闪存：read/write/erase操作
2. OFTL直接访问各页面的页面元数据
3. OFTL向文件系统开放字节单元访问接口→使得文件系统基于对象且不必管理存储只需管理命名空间



嵌入式FTL→需要嵌入式处理器提供可靠的计算能力/大容量DRAM



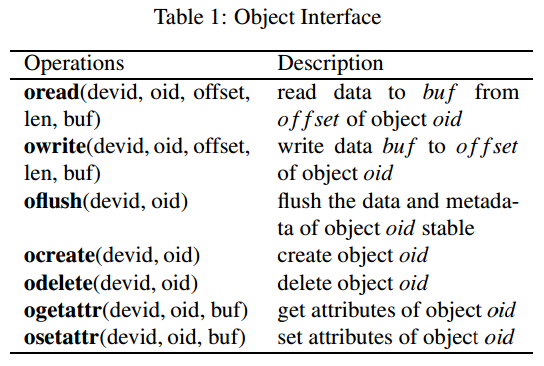
文件系统与FTL间块接口较窄→限制文件系统/FTL的优化

文件系统：块接口较窄→隐藏文件语义→限制智能存储管理

块接口较窄→闪存特性对文件系统不透明→错过优化机会

②OFTL接口优点

文件系统：字节单元读写接口→直接以字节大小访问OFTL



1. oread/owrite中offset/len均以字节为单元传入OFTL

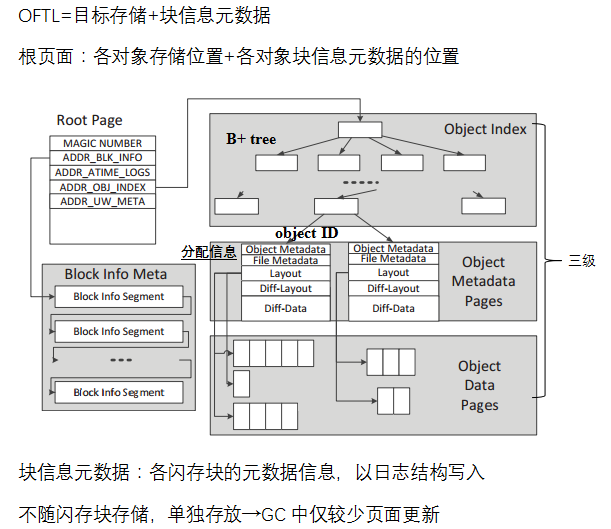
→OFTL知道由应用传来的访问的具体大小

→使得“压缩小更新到较少页面”成为可能

1. 各目标上的操作标有oid→OFTL能够知道访问页面的数据类型

→OFTL利用对象语义来聚类更新相关数据

③OFTL布局



1. **在页面元数据中保存逆索引和事务信息→缓慢刷新索引/消除日志**

**→减少索引元数据持久性/日志的成本**

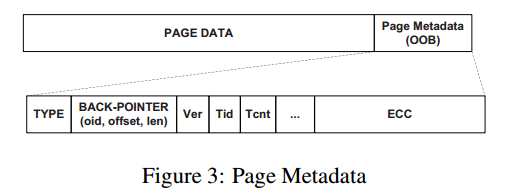
①索引元数据：（1）目标元数据页中数据布局的指针，指向数据页

（2）目标索引的指针，指向目标元数据页

索引持久性：数据丢失/数据不一致时，索引元数据应同步写入存储

设备→频繁的索引持久性造成严重的写放大

→使用延迟索引技术→保持一致性的同时，降低索引持久性的频率



②OFTL中后向指针(oid,offset,len)，分两种：

1. 用于数据页，反向索引对象元数据

有效参数：①oid：目标id ②offset：目标的逻辑页偏移

③len：页面的有效数据长度

1. 用于目标元数据页，反向索引目标索引

有效参数：①oid：目标id

写入后向指针后，首先设置类型以明确反向索引的类型→利用特定类型区别索引持久性与页面更新

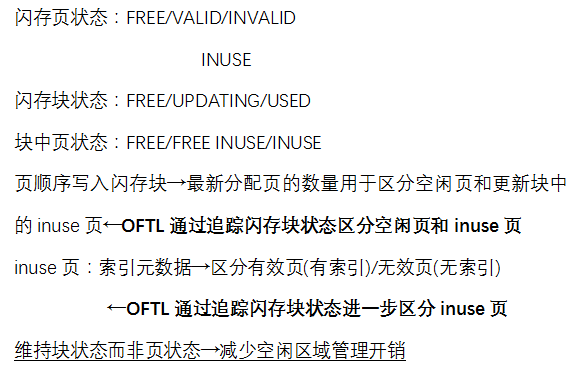
③同步索引元数据需要扫描反向索引，为减少扫描时间，使用**更新窗口**追踪未完成索引元数据同步的闪存块

1. 对于未同步索引元数据的闪存块：更新窗口**描述**一组该类闪存块→闪存块被预分配且其地址写于名为“更新窗口元数据”的闪存块中，该块持久写入闪存→从更新窗口分配闪存块并通过存储器索引引用，但该索引不持久写入 ←**“延迟持久性”**
2. 对于已同步索引元数据的闪存块：检查点进程定期的从更新窗口元数据中删除它们的地址，并在需要时预分配新的闪存块集合并在更新窗口中添加它们的地址

GC不能用于更新窗口的闪存块→事务信息用于完成度的原子确认

→消除日志

1. **采用粗粒度块状态跟踪页面状态→降低管理空闲空间的开销**

****

**4.字节单元接口→OFTL利用元数据压缩/共同定位局部页面更新**

**→减少页面更新的数量**

**5.实现基于OFTL的系统(OFSS)+不同类型工作负载→现存文件系统写放大的急剧减少**

**[2]**

File Defragmentation Scheme for a Log-Structured File System

日志结构文件系统的文件碎片整理方案

LFS：仅追加写机制→使大部分写入为顺序写入→避免随机写→解决闪存缺点

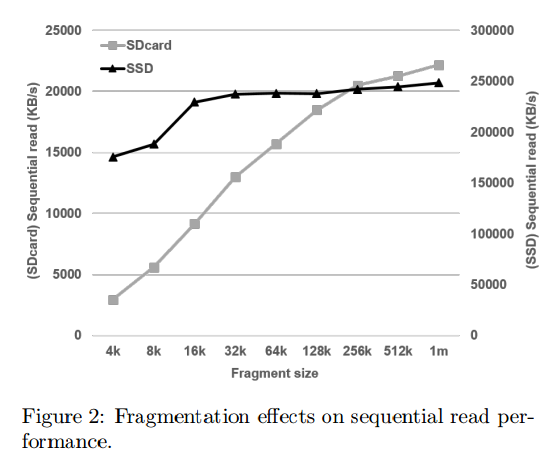
LFS缺点：文件碎片化问题→降低文件系统的各方面性能

文件碎片管理问题降低读性能的原因：

①块请求表示为连续块的列表→文件碎片化拆分块请求→干扰有效块I/O→降低I/O性能

②闪存预取功能：FTL预先加载后续数据→提高读性能

预取数据取决于LBA→文件碎片化导致FTL预取无效数据(文件被分离为多个碎片)→降低读性能



清洁线程的详细步骤：

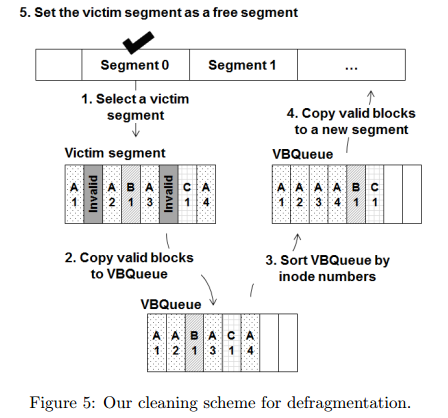
①根据受害者选择机制(如贪心或成本效益策略)选择受害段

②查看受害段中的各块是否有效

③探测到有效块后将有效块通过主存拷贝至新段

④对受害段中所有块重复2-4步

LFS现存清理工作不考虑受害块的顺序→无法消除文件碎片化问题



本文针对文件碎片化提出：LFS的新型清理机制

新型清洁机制：将段中受害块重新排列

1. 通过受害者选择机制选择受害者并查看段中块是否有效
2. 探测到有效块则将其加载到主存的VBQueue中，即将其延迟到新段
3. 已将全部有效块加载到VBQueue中后，通过节点数字整理有效块并拷贝到新段→整理后有效块与同一文件的其他块是连续的(因各文件具有独一无二的节点数字)

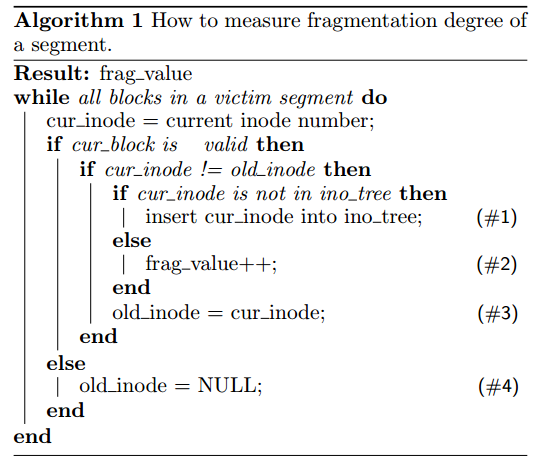
→消除文件碎片化

→整理过程无写入操作→开销极小

现存受害者选择机制不能实现新型清洁操作→提出碎片病毒受害者选择机制实现新型清洁操作

碎片病毒受害者选择机制：利用各段的碎片化程度来选择受害段

(碎片化程度=选择过程中有效块的数量)



当前块有效→当前块节点与前一块节点对比

→若相同→两块属于同一文件且LBA连续

→若不同→检测当前块节点是否包含于节点树中

→若包含→检查过的块已有该文件的另一块→文件碎片→碎片化程度+1

→若不包含→该块是该文件中出现的第一块→未碎片化