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

Abstract

高性能/低能耗/低成本→闪存成为企业/嵌入式系统常用的存储设备

**但** 仍存在问题：闪存的寿命

且 多级单元(MLC)的采用→存储密度增加 使得问题恶化

现有解决的方法：克服磨损均衡/数据减少

发现：文件系统的机制(日志/元数据同步/页面对齐更新)→引起额外写操作/加剧闪存的磨损 ←“文件系统的写放大问题”

解决方法：设计一种基于对象的闪存转换层(OFTL)

+机制与闪存共同设计

1. 利用页面元数据→OFTL实现索引元数据的延迟持久性

/OFTL在保证一致性的同时消除日志

1. 保持粗粒度块状态→减少管理持久空闲空间的开销
2. 通过字节单元访问接口→OFTL利用元数据压缩/共同定位小更新

→进一步减少更新

**OFSS：基于OFTL的系统**

1 Introduction

闪存的主流设计演变：单级单元(SLC)→多级/三级单元(MLC/TLC)

使得 容量增加/每比特成本降低

→闪存在企业和嵌入式存储系统中广泛应用

**但** 闪存密度的增加需要各单元内更精细的电压阶跃

→对泄漏/噪声干扰的耐受性差→降低了闪存的可靠性和寿命

常用解决方法：磨损均衡/数据减少→延长闪存寿命

1. 磨损均衡：P/E操作作用于闪存块→使块磨损均衡
2. 数据减少：共同作用于FTL/文件系统

①FTL：引入重复数据删除/压缩技术→避免冗余数据的更新

②文件系统：通过尾部打包/数据压缩→减少更新

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→无法解决持久性问题**

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

1. 块接口较窄/分层设计原则→文件系统和设备间不透明

发现

1. 各页面有元数据区域(即OOB区域)

用于保存 ①用于索引元数据延迟持久性的反向索引

②提供写入原子性的事务信息

1. 保持粗粒度块状态→减少管理持久空闲空间的开销
2. 随机读性能的提升

→能够压缩局部页面更新而不产生明显的随机读取惩罚

**则 可利用闪存特性来减轻耐久性问题**

减少写放大的解决方法：

设计一种基于对象的闪存转换层(OFTL) (OFTL将存储管理卸载到对象进行存储即存储在OFTL中)+系统与闪存共同设计

**本文贡献**

1. **OFTL**
2. **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. **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**
2. **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**

1. **实现OFSS+不同类型工作负载→现存文件系统写放大的急剧减少**

2 Background  
2.1 Flash Memory Basics

可用于文件系统设计的闪存特性：无覆盖写属性/耐久性限制

1. 无覆盖写属性：写前擦除
2. 耐久性限制：P/E操作次数有限

2.2 Architectures of Flash-based Storage Systems

嵌入式系统中，闪存不支持块接口且直接由文件系统管理，文件系统实现映射/GC/磨损均衡，为闪存定制 →“闪存文件系统” **图(a)**

FTL也可实现于设备固件(SSD)→提供块接口 **图(b)**

上述均为嵌入式FTL→需要嵌入式处理器提供可靠的计算能力

/大容量DRAM

FTL in software(VSL)：共享主机CPU周期和主存储器容量 **图(c)**

→为文件系统的优化提供机会 且 性能优于嵌入式FTL

但 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

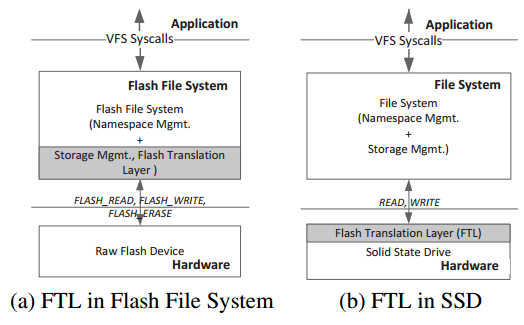
块接口较窄→闪存特性对文件系统不透明→错过优化机会

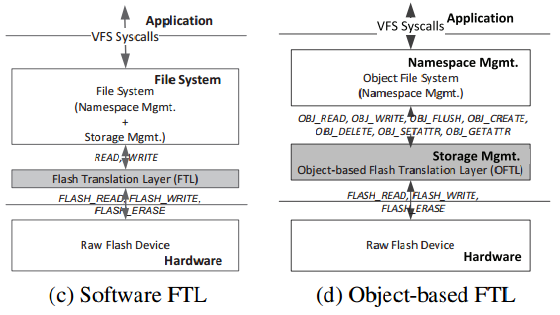
OFTL→更好的协作文件系统和闪存 **图(d)**

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



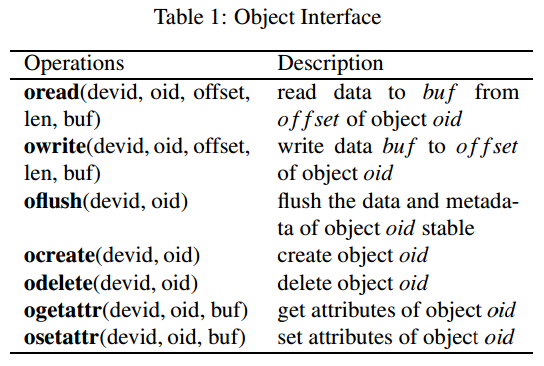


3 Object-based Flash Translation Layer

OFTL映射转换：各对象逻辑偏移→闪存页地址

**OFTL接口**

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

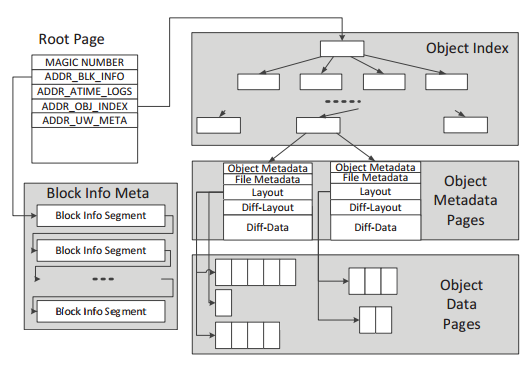
→OFTL利用对象语义来聚类更新相关数据

1. 数据页面附有类型语义→OFTL区分索引页面与数据页面→使得在页面元数据中保留辅助元数据→延迟索引

**数据组织**

OFTL=目标存储+块信息元数据

根页面：各对象存储位置+各对象块信息元数据的位置



三级

**分配信息**

**object ID**

**B+ tree**

块信息元数据：各闪存块的元数据信息，以日志结构写入

不随闪存块存储，单独存放→GC中仅较少页面更新

块信息元数据=闪存块状态+无效页面的数量+各闪存块的擦除次数

32bits 2bits 10bits 20bits

闪存块状态：FREE/UPDATING/USED

闪存页状态：FREE/VALID/INVALID

4 System Co-design with Flash Memory

OFTL利用闪存特性的三个技术：

1. Backpointer-Assisted Lazy Indexing：有效保持数据与元数据间的一致性
2. Coarse-Grained Block State Management：降低状态写的频率
3. Compacted Update：分摊多个未对齐页面写入的成本
   1. Backpointer-Assisted Lazy Indexing

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

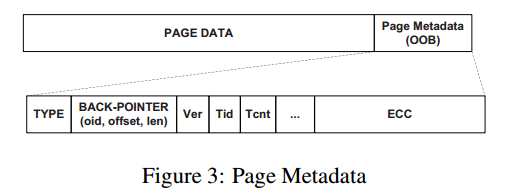
数据丢失/数据不一致时，索引元数据应同步写入存储设备

→“索引持久性”→ 频繁的索引持久性造成严重的写放大

(eg.指针大小为8字节，但索引持久性更新整个页面，即4KB或更大)

**希望**：保持一致性的同时，降低索引持久性的频率

**解决方法**：延迟索引技术(在索引元数据的延迟持久性中的各索引页的页面元数据中采用后向指针，该指针类型特定)



OFTL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→特定类型的后向指针作为反向索引并将索引持久性与页面更新分离

系统错误时重建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**追踪近期分配的未完成索引元数据持久性的闪存块

更新窗口由检查点进程维持

更新窗口**触发条件**：其中空闲页面数量低于阈值&需要窗口扩展

因其可能不会被索引引用→更新窗口**描述**一组其反向索引需要在故障后确认的闪存块→(这些)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”

对于已经持久写入相应索引的那些闪存块，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

→→故障后，只需读取更新窗口元数据中的地址对应的块并确认其能被索引引用

更新窗口提供多页面更新的原子性(完整性←事物是不可分割的整体)

页面元数据中：事务信息tid/tcnt

tid：各写入操作在更新窗口中的id（该id独一无二）

tcnt：更新的页面数量

更新的所有页面中，仅一个页面设置tcnt参数，其余页面tcnt=0

tcnt用于确认系统故障后操作是否完成

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

系统故障后重启→扫描更新窗口中的目标数据页→首先核对事务信息以确保写入操作的完成→若某一写入未完成→整个页面的写入都被丢弃←确保原子性

→若目标布局在系统故障前未被平稳写入→读取后向指针以更新目标布局

（因 所有更新都位于来自上一个检查点的更新窗口→通过重建来自当前更新窗口的扫描页面中的后向指针→将目标布局更新至最新版）

重算所有有效数据大小→更新文件大小元数据

其他描述性元数据(修改时间/访问控制列表)可能在意外崩溃后丢失，但不会影响系统一致性←不考虑更新

扫描目标元数据页的当前更新窗口→更新目标索引

页面元数据的辅助信息/更新窗口→降低索引持久性频率/保证写入原子性以消除日志

**一致性和耐久性讨论**

延迟索引技术中，除索引指针和大小外，其他元数据会在系统崩溃后丢失→元数据版本过时→损伤元数据更新的耐久性

选择文件系统/应用程序→确认时间→通过显式同步→保证元数据耐久性

延迟索引技术→保证索引和大小能在恢复过程中重建

4.2 Coarse-Grained Block State Maintenance

闪存页状态：FREE/VALID/INVALID

INUSE

闪存块状态：FREE/UPDATING/USED

块中页状态：FREE/FREE INUSE/INUSE

页顺序写入闪存块→最新分配页的数量用于区分空闲页和更新块中的inuse页←**OFTL通过追踪闪存块状态区分空闲页和inuse页**

inuse页：索引元数据→区分有效页(有索引)/无效页(无索引)

←**OFTL通过追踪闪存块状态进一步区分inuse页**

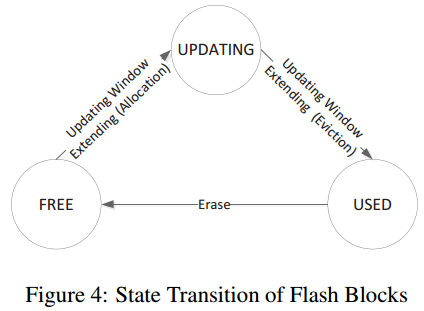
维持块状态而非页状态→减少空闲区域管理开销

(如图4)

**仅**在 闪存块分配给/被逐出更新窗口→执行闪存块状态的持久性

状态持久性的满足条件：

1. 持久空闲块的集合是实际空闲块集合的子集
2. 持久无效页的数量比无效页实际数量少



条件1→分配时执行持久性→部分实际空闲块不属于持久空闲块→该部分块可能会被视为非空闲→导致错误分配

解决方法：FREE→UPDATING快速刷新/**降低USED→FREE状态持久性**

条件2**降低无效页数量持久性**

无效页数量→选择逐出闪存块/核对GC时删除的有效页数量

有效页数量达标→其余页面无效，不需再核对无效页→停止对于逐出块中无效页的删除

综上 降低元数据持久性频率→减少开销

4.3 Compacted Update

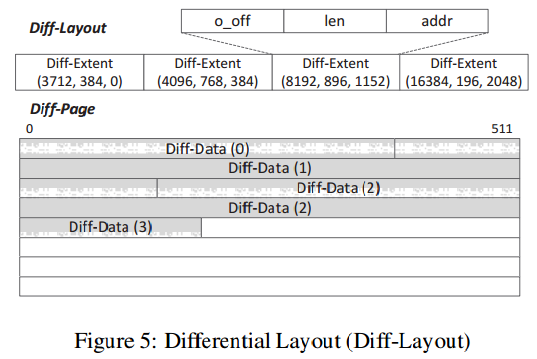
OFTL的字节单元访问接口→识别局部页面更新

局部页面更新：仅更新某闪存页的某部分(小于单个闪存页大小的小更新/大更新的头或尾)

压缩更新技术：将具有相同目标和共同定位的局部页面更新与其目标元数据页面共同压缩，以减少更新页面

**局部页面更新**

OFTL的局部页面更新被压缩并插入到差异页面(diff-pages)中



diff-page中各数据段：diff-data

diff-layout：各目标有一个diff-layout，是单个目标的全部diff-extent的集合

diff-data索引：diff-extent 三元参数<o\_off,len,addr>，以diff-layout中目标偏移量的升序保存diff-extent

①o\_off：diff-data的目标偏移量 ②len：diff-data的目标长度

③addr：diff-page中diff-data的偏移量

局部页面更新插入diff-pages，由diff-layout索引

完整页面更新直接写入目标页面，由layout(不是diff-layout)索引

layout：某目标的全部范围，包括起始地址/完整页面的长度对

diff-layout与layout记录于目标元数据页中

压缩更新技术中读/写/合并操作：

1. 写入：写入数据被分为页面，有局部页面更新/完整页面更新，局部页面更新于diff-pages中更新

局部更新常取代完整页面→完整页面更新通过更新layout

→置对应diff-data无效/移除其diff-extent→引用完整页面

1. 合并：diff-pages全满时需要合并操作

扫描diff-extents→选择diff-pages中消耗最多空间的逐出逻辑页

→读取该逐出逻辑页的目标页面→将其与diff-data合并

→合并页面写入新的目标页面→移除对应diff-data/diff-extent

1. 读取：

核对diff-extents→若满足diff-data→读取diff-data到buffer

→若不满足diff-data→读取目标页面→与任意现存diff-data合并

**更新共同定位**

通常，各目标元数据大小小于页面大小→不再压缩多个目标的元数据→OFTL将diff-page/目标元数据页共同定位

因 数据更新后更新元数据(文件大小/修改时间)→diff-data的共同定位和元数据页能减少一个页面的写入

diff-page大小：小于页面大小，取决于目标元数据的大小变化，用于确认diff-page是否全满

diff-page大小=闪存页大小-元数据大小

diff-page全满→触发合并操作→选择一些diff-data→将它们与目标数据页合并→局部页面更新的开销被压缩和共同定位分摊