CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

南京工業大學
NANJING TECH UNIVERSITY

F2FS Large Folios 新进展

针对压缩文件和私有状态的原创设计与实现

演讲人:赵南哲

汇报提纲

1. 社区进展回顾
2. 问题背景
3. 设计方案
4. 未来进展规划

F2FS Folios进展回顾

- F2FS社区large folios工作现状：
  - Matthew Wilcox 连续发送三次folio转换补丁，f2fs从接口层面已经基本全面支持folios结构体。
  - 2024年18届CLK,vivo已经提出过f2fs上使用iomap框架以支持large folios的实践方案。
- F2FS社区large folios尚未完成部分：
  - F2FS对folios的支持还比较初级,目前只是停留在接口层面,只能使用0阶folio。
  - 目前公开过的large folios支持方案里,暂未包含压缩文件。
  - F2FS的page private标志位与iomap_folio_state冲突。

39. **[f2fs-dev] [PATCH 00/27] f2fs folio conversions for v6.15**
    - by Matthew Wilcox (Oracle) @ 2025-02-18 5:51 UTC [12%]

27. **[f2fs-dev] [PATCH 000/153] f2fs folio conversions for 6.16**
    - by Matthew Wilcox (Oracle) @ 2025-03-31 20:10 UTC [10%]

8. **[f2fs-dev] [PATCH 00/60] f2fs folio conversions for 6.17**
    - by Matthew Wilcox (Oracle) @ 2025-07-08 17:02 UTC [11%]

汇报提纲

1. 社区进展回顾
2. **问题背景**
3. 设计方案
4. 未来进展规划

为什么要适配iomap_folio_state

- iomap_folio_state简介：
  - 自从linux 6.6起，内核iomap框架引入了iomap_folio_state 它能够追踪folio中具体每个文件系统块的uptodate和dirty状态，用于避免large folios写回磁盘造成的**写放大问题。**
  - 同时,iomap_folio_state具备read/write_bytes_pending 字段 能记录folio并发处于io中的所有子区间的字节数。这对于folio的io状态生命周期**至关重要**,因为只有**当整个folio所有子区间io结束时候才应该调用folio_end_read/writeback。**
- 为什么F2FS支持large folios需要它：
  - F2FS**不具备buffered head/subpage这样原生的逐块追踪数据结构**,io路径代码全部基于单page索引与单文件系统块一对一映射假设,因此原生代码**无法追踪**folio中所有并发io子区间。
  - 逐块脏状态追踪对F2FS来说至关重要,因为large folios的写放大对于闪存友好的F2FS来说是**不可接受**的。

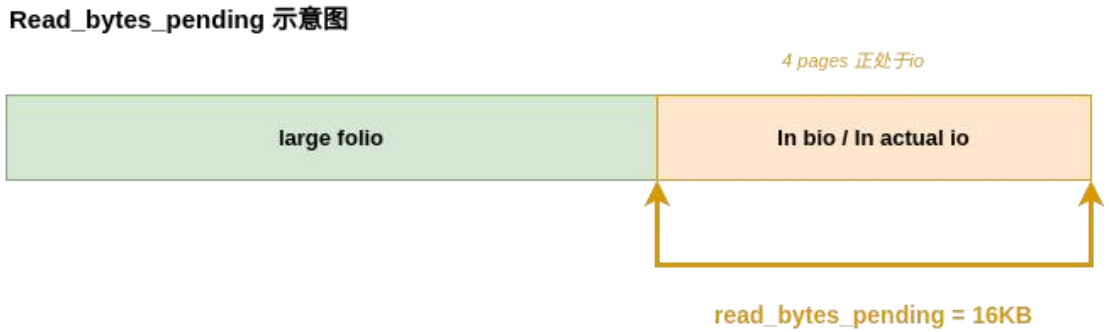
为什么要适配iomap_folio_state

```c
/*
 * Structure allocated for each folio to track per-block uptodate, dirty state
 * and I/O completions.
 */

struct iomap_folio_state {
	spinlock_t		state_lock;
	unsigned int		read_bytes_pending;
	atomic_t		write_bytes_pending;

	/*
	 * Each block has two bits in this bitmap:
	 * Bits [0..blocks_per_folio) has the uptodate status.
	 * Bits [b_p_f...(2*b_p_f))   has the dirty status.
	 */
	unsigned long		state[];
};
```

位图中的每个 bit 对应 folio 中的一个 page，1 表示脏页（红色），0 表示干净页（绿色）



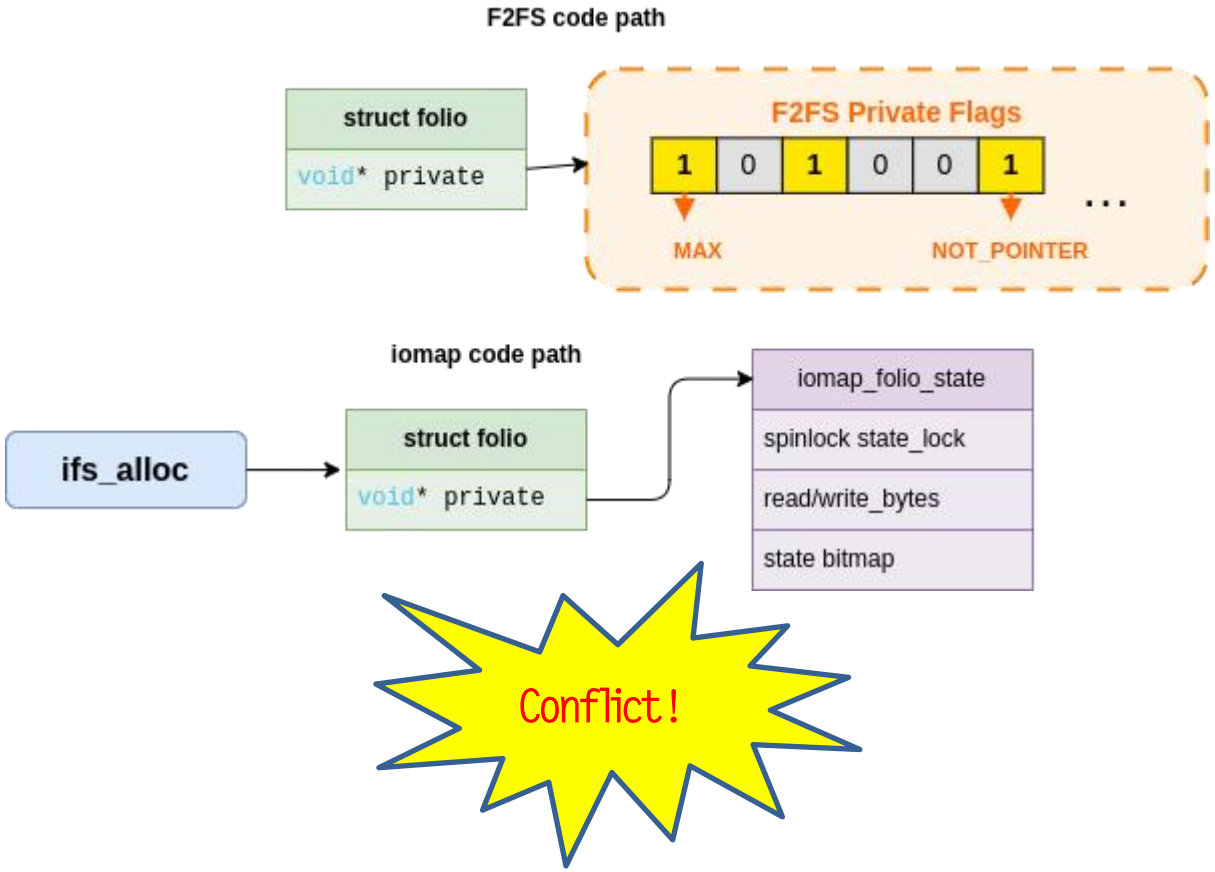
read_bytes_pending 跟踪folio正在处于io状态的字节数

核心冲突:F2FS Page Private标志

- F2FS page private 字段:
  - F2FS利用内核分配的地址空间低三位总是与8对齐,末尾三位总是为0,以page/ folio->private最后一位设置PAGE_PRIVATE_NOT_POINTER表明启用F2FS私有标志位扩展。
  - 当该扩展启用时，F2FS会利用private字段的低5位存放标志位，此时剩余位数已经无法再容纳一个内核指针地址。
- iomap_folio_state分配方式:
  - iomap框架总假定folio->private为指针值,如果folio->private为空正常分配iomap_folio_state。如果有值则无论其为何值总是认为iomap_folio_state总存在。

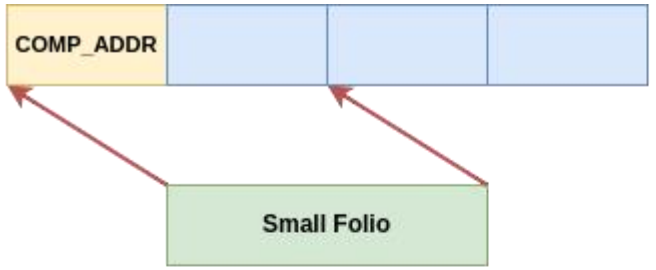
二者竞争使用folio->private标志位,产生冲突!

  - 同时,iomap框架现有数据结构的private成员全部只有栈上局部变量生命周期,而F2FS private字段生命周期存在于整个folio存活阶段,无法解决F2FS问题。



压缩文件的挑战:跨压缩/非压缩簇

- F2FS支持文件部分压缩,压缩簇page数量以4的整数倍(下面假设为4)作为固定单位。同时可以存在非压缩部分。
- 但folio在page cache中的阶数大小是不固定的。可大可小。大folio可以同时跨越压缩/非压缩簇。小folio会可能完全覆盖压缩簇的某个部分。
- 因为folio可以同时跨越压缩,非压缩部分的缘故,压缩簇部分对应的folio的子区间同样需要加上读写io字节计数追踪,以在正确时机结束folio的生命状态。



Small order folio entirely inside cluster

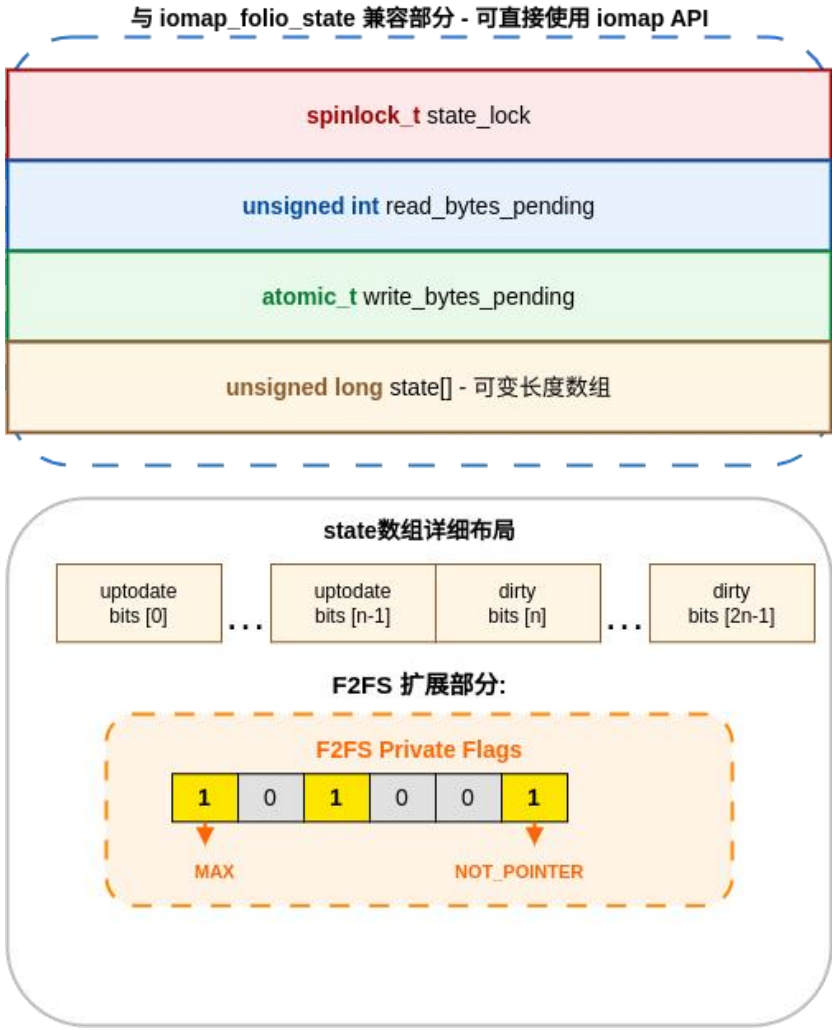
Large folio can cross multi compressed/uncompressed clusters

汇报提纲

1. 社区进展回顾
2. 问题背景
3. **设计方案**
4. 未来进展规划

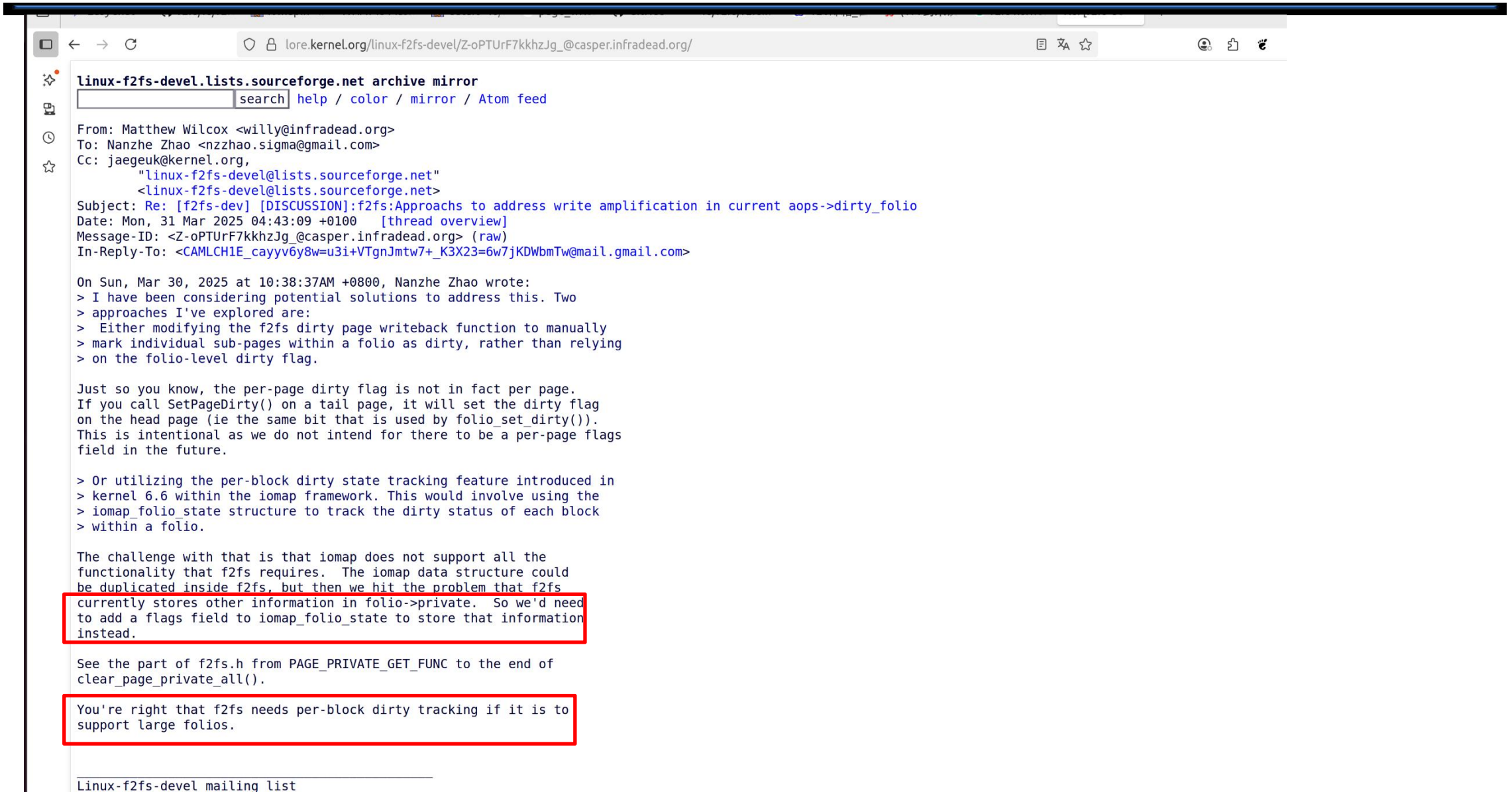
扩展iomap_folio_state

- **字段扩展:**通过扩展柔性数组state尾部字段,可使改进后iomap_folio_state具备F2FS page private字段感知。
- **魔数区分:**将预读后归零的read_bytes_pending初始化为magic number,精准区分原版和f2fs自己的iomap_folio_state。
- **布局兼容:**结构体靠前的字段和原iomap_folio_state完全一致,可以直接兼容iomap的api。



扩展iomap_folio_state设计图

扩展iomap_folio_state



lore.kernel.org/linux-f2fs-devel/Z-oPTUrF7kkhzJg_@casper.infradead.org/

linux-f2fs-devel.lists.sourceforge.net archive mirror

search help / color / mirror / Atom feed

From: Matthew Wilcox <willy@infradead.org>
To: Nanzhe Zhao <nzzhao.sigma@gmail.com>
Cc: jaegeuk@kernel.org,
"linux-f2fs-devel@lists.sourceforge.net"
<linux-f2fs-devel@lists.sourceforge.net>
Subject: Re: [f2fs-dev] [DISCUSSION]:f2fs:Approachs to address write amplification in current aops->dirty_folio
Date: Mon, 31 Mar 2025 04:43:09 +0100 [thread overview]
Message-ID: <Z-oPTUrF7kkhzJg_@casper.infradead.org> (raw)
In-Reply-To: <CAMLCH1E_cayyv6y8w=u3i+VTgnJmtw7+_K3X23=6w7jKDWbmTw@mail.gmail.com>

On Sun, Mar 30, 2025 at 10:38:37AM +0800, Nanzhe Zhao wrote:
> I have been considering potential solutions to address this. Two
> approaches I've explored are:
> Either modifying the f2fs dirty page writeback function to manually
> mark individual sub-pages within a folio as dirty, rather than relying
> on the folio-level dirty flag.

Just so you know, the per-page dirty flag is not in fact per page.
If you call SetPageDirty() on a tail page, it will set the dirty flag
on the head page (ie the same bit that is used by folio_set_dirty()).
This is intentional as we do not intend for there to be a per-page flags
field in the future.

> Or utilizing the per-block dirty state tracking feature introduced in
> kernel 6.6 within the iomap framework. This would involve using the
> iomap_folio_state structure to track the dirty status of each block
> within a folio.

The challenge with that is that iomap does not support all the
functionality that f2fs requires. The iomap data structure could
be duplicated inside f2fs, but then we hit the problem that f2fs
currently stores other information in folio->private. So we'd need
to add a flags field to iomap_folio_state to store that information
instead.

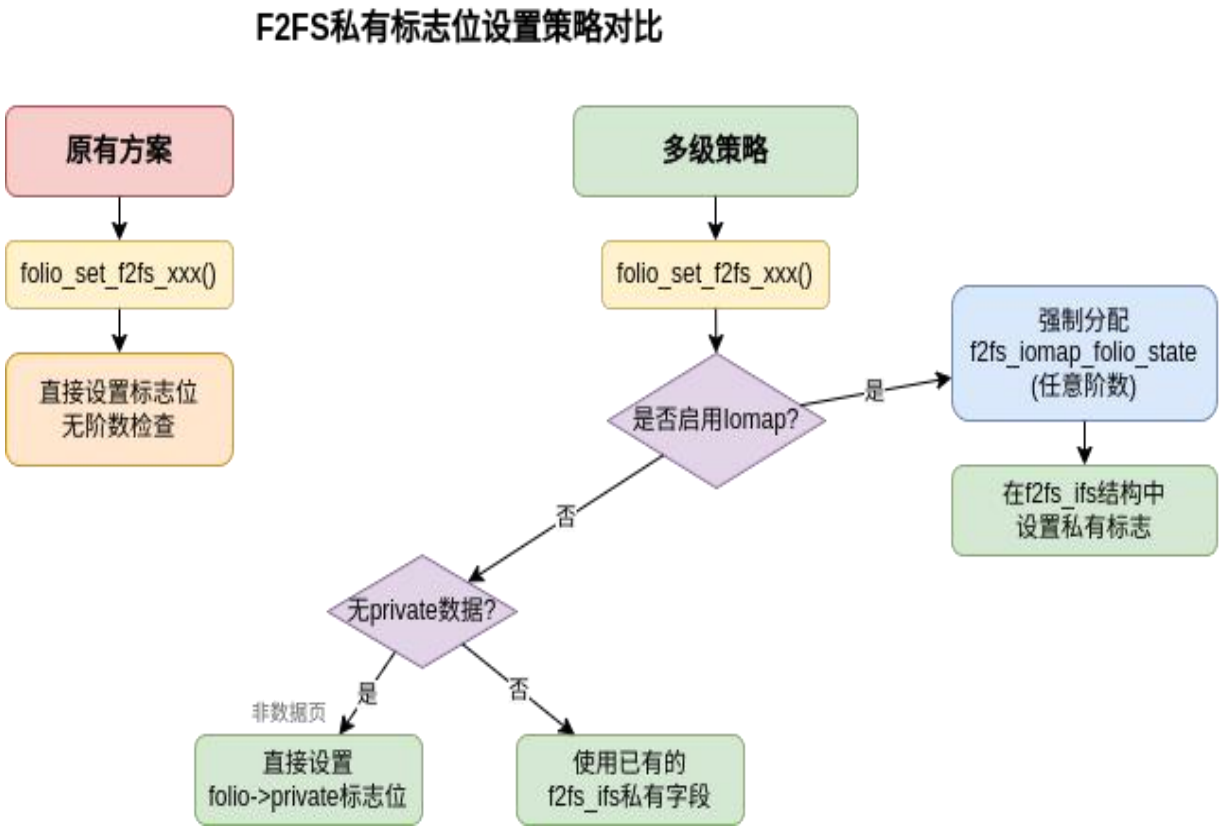
See the part of f2fs.h from PAGE_PRIVATE_GET_FUNC to the end of
clear_page_private_all().

You're right that f2fs needs per-block dirty tracking if it is to
support large folios.

_____
Linux-f2fs-devel mailing list

扩展folio private帮助函数-以set为例

- 按种类分配:为文件页分配f2fs_iomap_folio_state后存入f2fs标志位,对于所有不适用iomap的inode(F2FS中所有的存放元数据的inode,压缩page cache inode)直接设置F2FS标志位。
- 强制分配:当folio_set_f2fs_xxx被调用的时候,0阶folio也会被强制分配一个f2fs_iomap_folio_state。因为不这么做的话会存在这样的情况:GC先为folio->private直接设置标志位,然后buffered write走iomap路径错误将folio->private识别为指针。

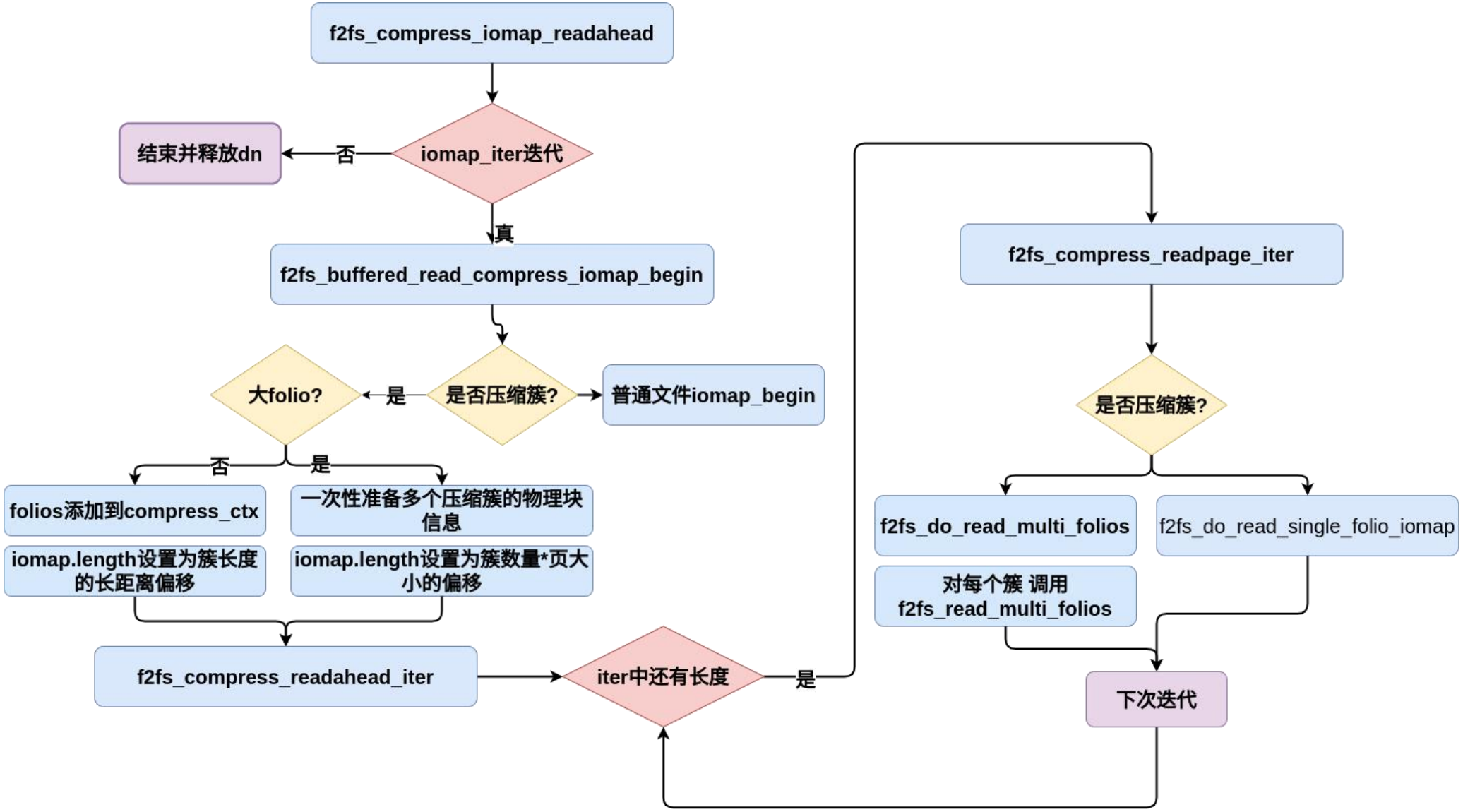


多级标志位设置策略设计图

压缩文件folio多簇读方案-迭代框架

- 压缩文件的io提交方式与iomap_readpage_iter不同,是压缩folio加入到bio之中而非page cache中的文件folio。
- 因而仿照iomap框架思路,定义f2fs_compress_readahead/readpage_iter函数,使用iomap_iter迭代循环。并仿照iomap_readpage_ctx,定义f2fs_readpage_ctx管控folio迭代状态,并加入扩展成员。
- 自定义f2fs_buffered_read_compress_iomap_begin,碰到压缩簇正常走压缩逻辑,碰到非压缩簇直接调用为普通文件使用的f2fs_buffered_read_iomap_begin。

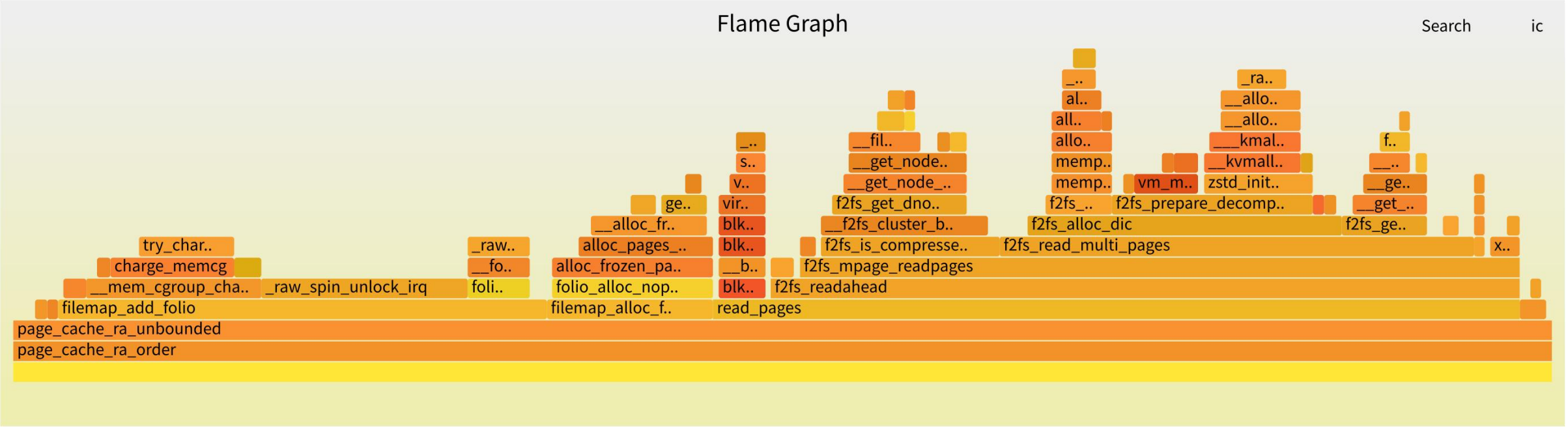
压缩文件folio多簇读方案-迭代框架



压缩文件folio多簇读方案-dn cache

- **性能瓶颈:**原压缩文件读代码判断压缩簇和查找压缩簇块地址全都要调用f2fs_get_dnode_of_data。大量的多级间接指针查找成为性能瓶中很大一部分占比。
- **dn缓存:**将第一个簇查找之后拿到的dnode_of_data缓存在f2fs_readpage_ctx,之后判断索引上相邻的簇是不是压缩簇或查找块地址只要递增ofs_in_node,从缓存的dn中查找即可。碰到跨dn的情况,就滚动释放原先的dn,用f2fs_get_dnode_of_data去取下一个。
- **资源生命周期管理:**缓存在f2fs_readpage_ctx中的dn的资源释放,不论是正常结束还是错误退出,由最外层的函数统一释放处理。

压缩文件folio多簇读方案-dn cache

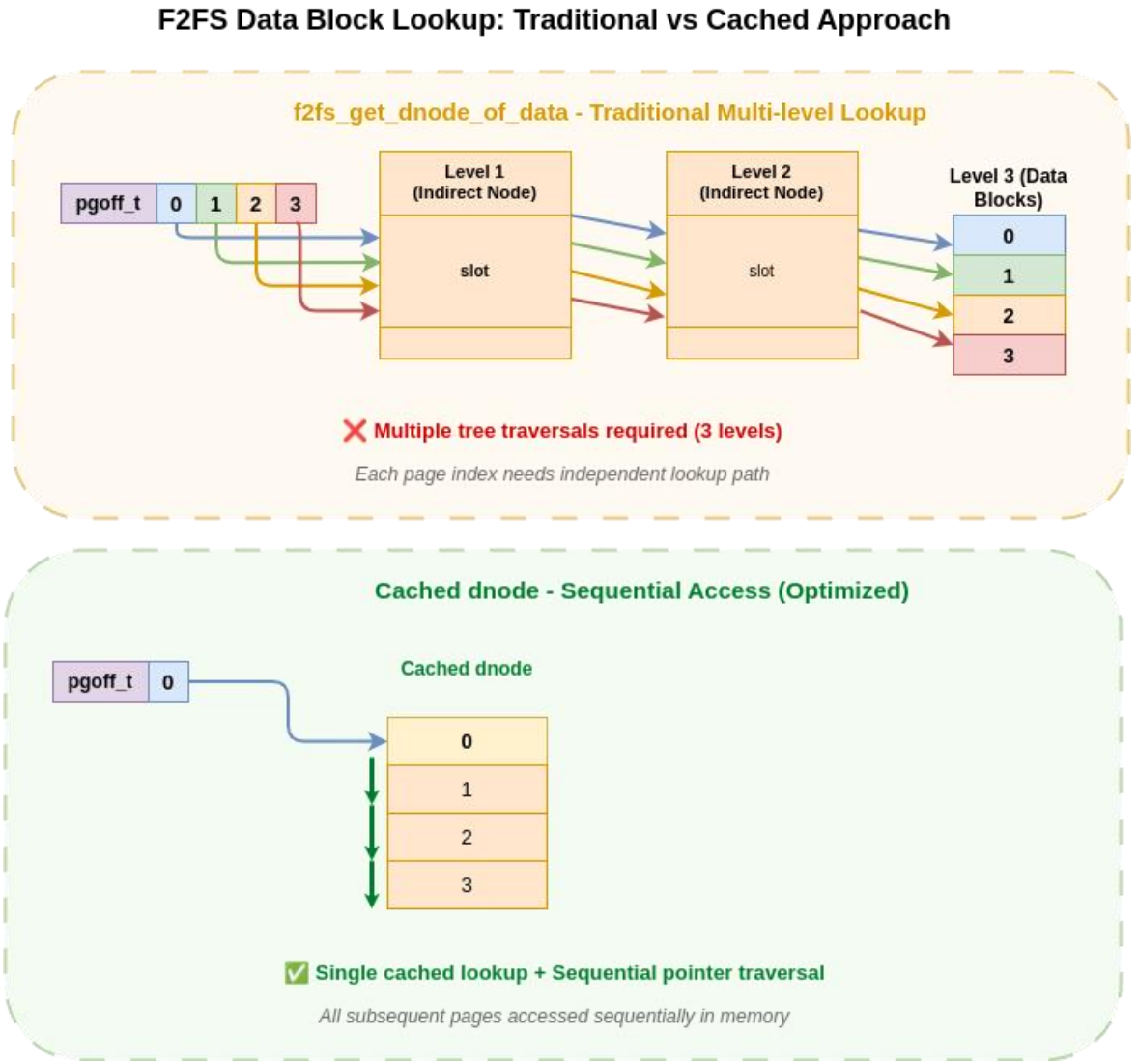


原压缩文件buffered read 火焰图

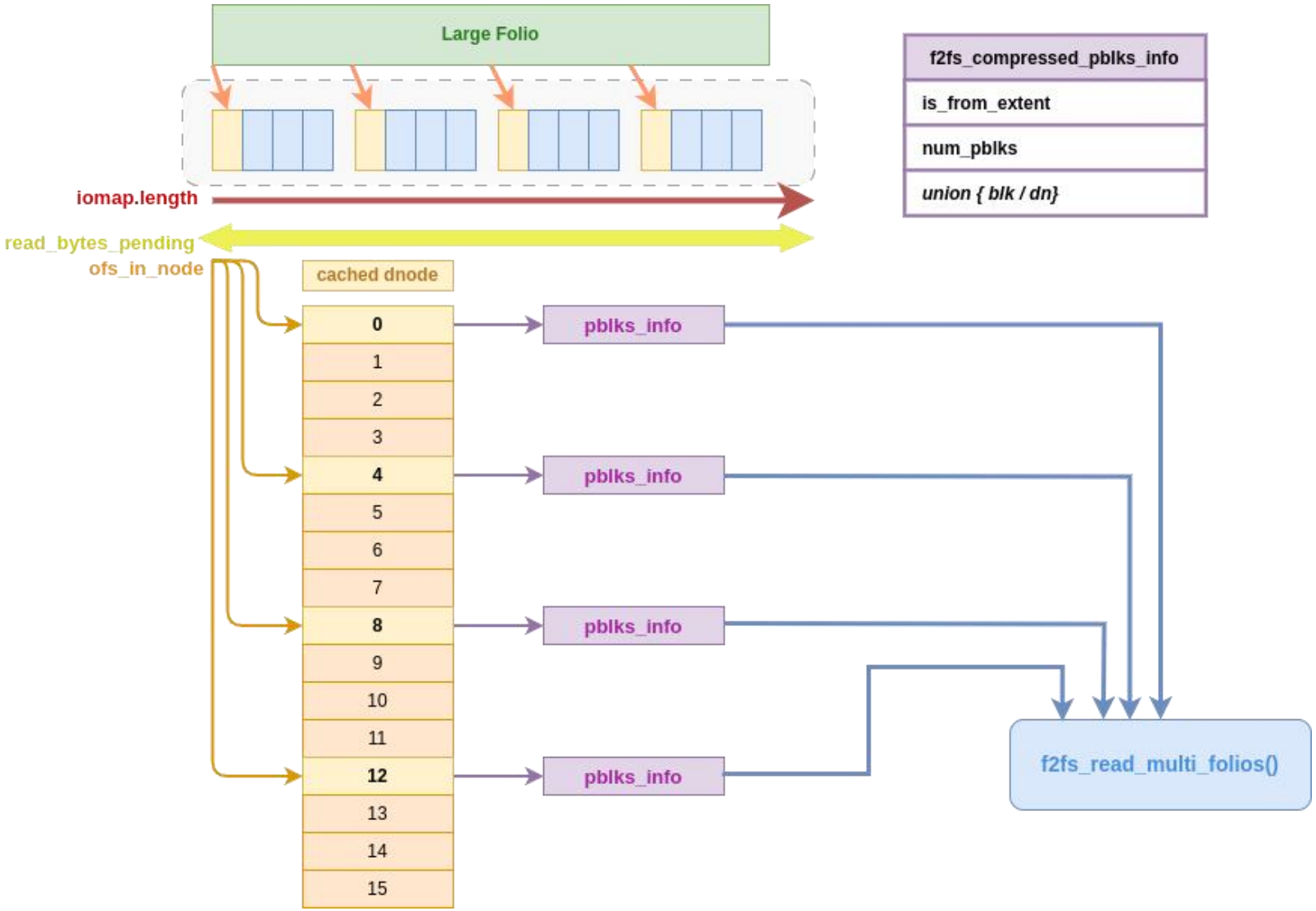
主要性能开销由filemap_add_folio,filemap_alloc_folio和readahead中dic的分配和f2fs_get_dnode_of_data组成。

压缩文件folio多簇读方案-dn cache

```
struct f2fs_readpage_ctx {
    struct folio        *cur_folio;
    bool                cur_folio_in_bio;
    struct bio          *bio;
    struct readahead_control *rac;
    unsigned int compressed_clusters;
    struct f2fs_compressed_pblks_info pblks_arr[MAX_PBLLKS_INFO_SIZE];
    struct dnode_of_data dn; /* cached dnode of data */
    bool has_put_dn;
    struct compress_ctx cc;
    bool cur_folio_in_compress_ctx;
};
```



压缩文件folio多簇读方案-dn cache



压缩文件folio多簇读方案:读字节拓展

- 定义函数f2fs_iomap_finish_folio_read，拓展使用f2fs_iomap_folio_state的read_bytes_pending字段。并在使其在非压缩部分下保持和和非压缩文件一样的读记账逻辑，而在压缩部分下引入簇视角读记账。当folio向压缩上下文添加子部分时将这部分字节数统计入该folio的read_bytes_pending,当压缩上下文存储的压缩簇已满并解压之后,遍历簇内所有folio的子部分，**分别扣减它们在簇中所有子部分对应的字节数。**

```
void f2fs_iomap_finish_folio_read(struct folio *folio, size_t off,size_t len, int error)
{
    struct f2fs_iomap_folio_state *fifs =folio->private;
        bool uptodate = !error;
        bool finished = true;
        if(folio_order(folio)>0&&fifs) {
            unsigned long flags;
            spin_lock_irqsave(&fifs->state_lock, flags);
            fifs->read_bytes_pending-=len;
            #ifdef CONFIG_F2FS_DEBUG_PRINT
            FUNC(print_rbp,folio);
            #endif
            if(!error) {
                uptodate=ifs_set_range_uptodate(folio, (struct iomap_folio_state*)fifs,off,len);
            }
            finished = (fifs->read_bytes_pending == 0||fifs->read_bytes_pending==F2FS_IFS_MAGIC);
            spin_unlock_irqrestore(&fifs->state_lock, flags);
            if(finished) {
                folio_end_read(folio, uptodate);
            }
        }
        else {
            folio_end_read(folio, !error);
        }
}
```

压缩文件folio多簇读方案:读字节拓展

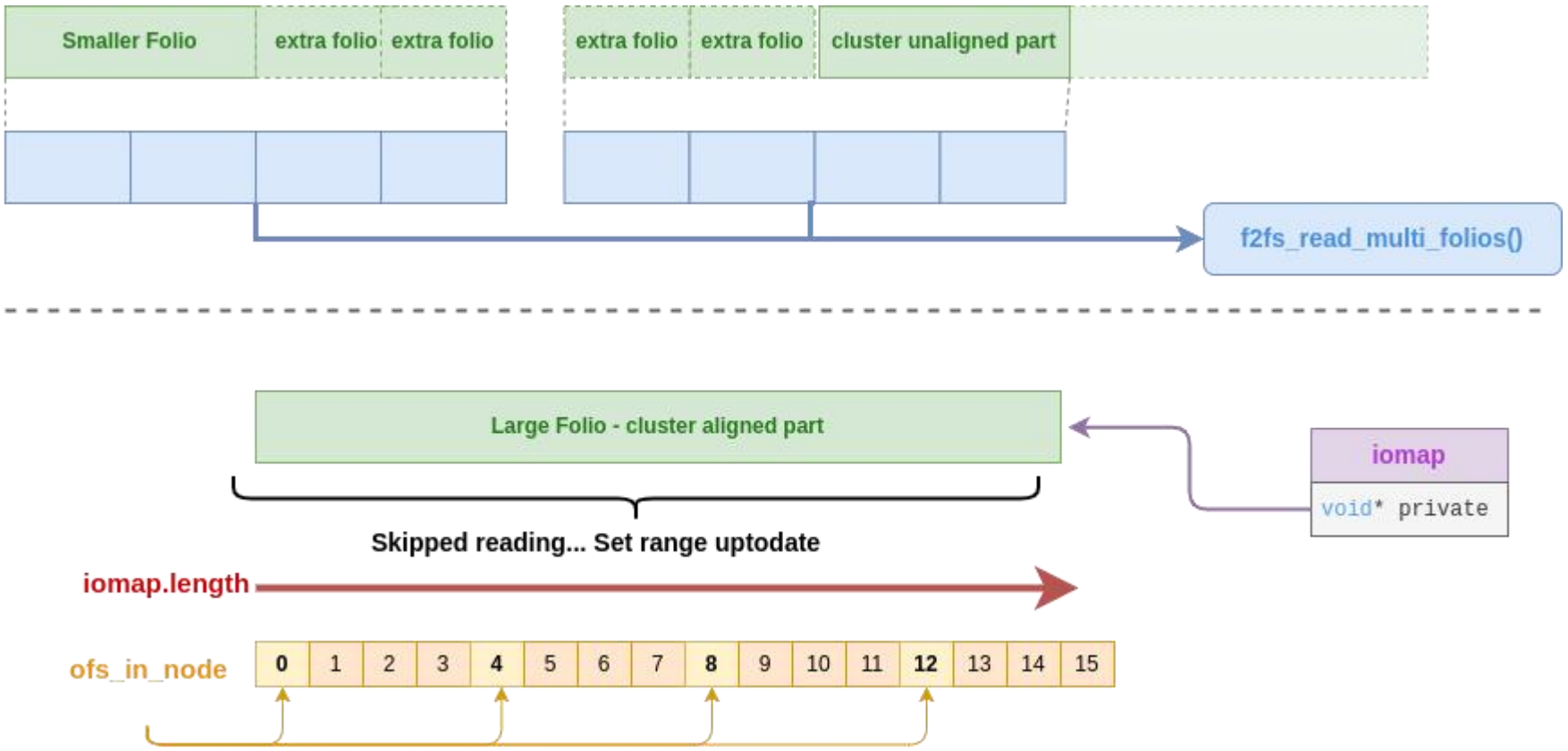
```
void f2fs_decompress_end_io(struct decompress_io_ctx *dic, bool failed,bool in_task)
{
    /* .... */
    int num_to_skip=0;
    for (int i = 0; i < dic->cluster_size; i+=num_to_skip) {
        struct folio *folio;
        num_to_skip=1;
        if (!dic->rpages[i])
            continue;
        folio = page_folio(dic->rpages[i]);
        while ((i + num_to_skip) < dic->cluster_size && dic->rpages[i + num_to_skip] &&
               page_folio(dic->rpages[i + num_to_skip]) == folio) {
            num_to_skip++;
        }
        struct f2fs_iomap_folio_state *fifs=folio->private;

    loff_t poff = (loff_t)(folio_page_idx(folio,dic->rpages[i])) << PAGE_SHIFT;
    if (failed) {
        if (folio_order(folio) > 0 && fifs) {
            f2fs_ifs_clear_range_uptodate(folio, fifs, poff, num_to_skip << PAGE_SHIFT);
            folio_clear_uptodate(folio);
            folio_unlock(folio);
            } else {
                folio_clear_uptodate(folio);
                folio_unlock(folio);
            }
        } else {
            f2fs_iomap_finish_folio_read(folio, poff, num_to_skip << PAGE_SHIFT, 0);
        }
    }
    /* ... */
}
```

压缩文件buffered_write:贪心策略

- 贪心策略:根据iomap_begin中传入的偏移和总写入字节,利用iomap_get_folio分配不超过这个范围的**最大folio。**。
- 边界簇填充： 若当前读取偏移开头不和簇索引对齐,将拿到的folio当前部分对应的整个簇进行一次读io,簇中空缺的部分分配新的0阶folio填满。同样若folio尾部也不能填满一个簇,做同样操作。
- 跳过中间簇:对于大folio所有全部覆盖的索引对其的中间簇,**全部跳过读io(因为是完全覆写)**,直接推进iomap->length并将该区间标记为uptodate。
- 同步读轮询:为read_bytes_pending增加1个bias,然后轮询等待其计数为魔数+1。**这样可同步等待完folio所有要io的部分,并且阻止folio_end_read被调用。**

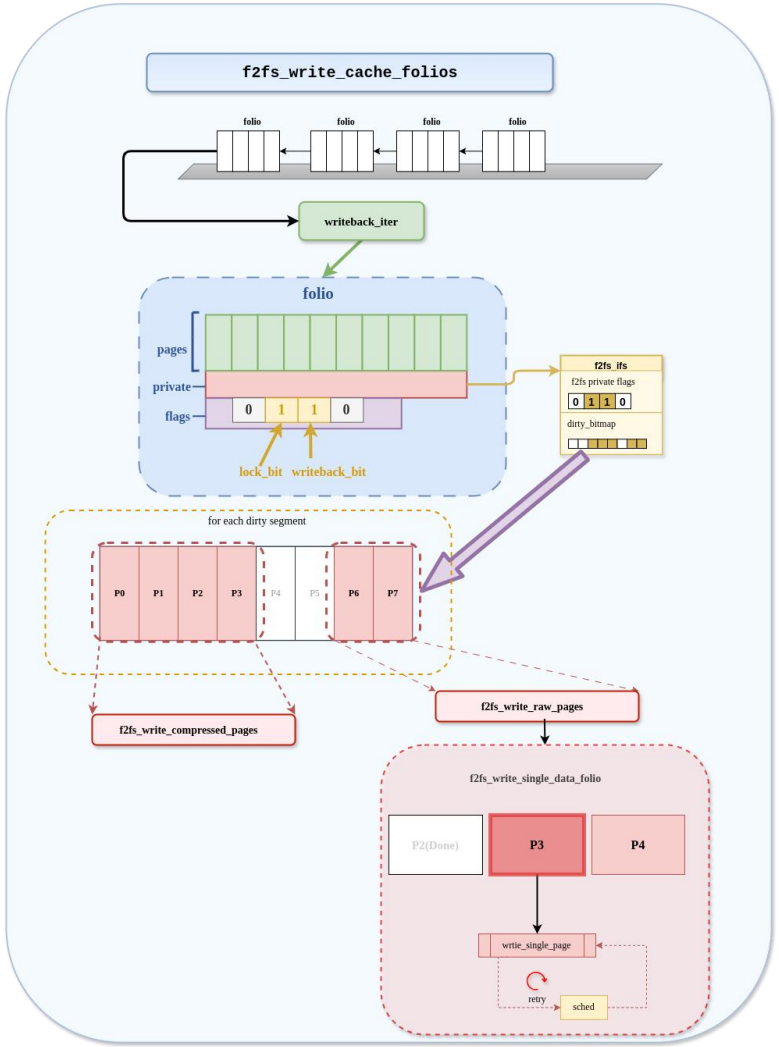
压缩文件buffered_write:贪心策略



压缩文件folios 脏页回写-迭代框架

- 压缩文件脏页回写的io方式同样需要借助于compressed folio。目前于iomap_writepages中的io方式假设差异性很大。
- 因而尝试仿照iomap思路,使用writeback_iter和iomap_find_dirty_range作为迭代folio脏区间的框架,并且限制每次返回的脏区间长度不超过一个簇。这之后将这个簇对应的folio部分添加到压缩上下文,当压缩上下文满了之后调用f2fs_write_multi_pages。
- 同buffered read,回写的时候我们也分别为folio的压缩簇/非压缩簇部分统一使用write_bytes_pending追踪folio即将处于io的字节数,并定义函数f2fs_iomap_finish_folio_write来扣减write_bytes_pending。

压缩文件folios 脏页回写–迭代框架



压缩文件folios 脏页回写-非解锁重试

- F2FS在脏页写回的时候,会先上page/folio锁然后以读模式上cp_rwsem锁。但是F2FS存在先以读模式上cp_rwsem锁然后再上page /folios锁的路径。两条路径倒序上锁,在checkpoint线程启动并以写模式上cp_rwsem锁后,会造成死锁(因为内核的rwsem默认实现是读写公平的)。因而F2FS将负责执行页面写回函数的f2fs_do_write_data_page内改为以非阻塞模式上cp_rwsem读锁,获取读锁失败之后会将当前page解锁并休眠一段时间后重试。
- 但是高阶folio中途解锁是危险行为。因而定义函数f2fs_write_single_folio,对folio的一段脏页区间对每个子page循环调用f2fs_do_write_data_page,当获取读锁失败后,保持folio锁不释放,之后从刚刚失败的folio内page索引处开始重试。

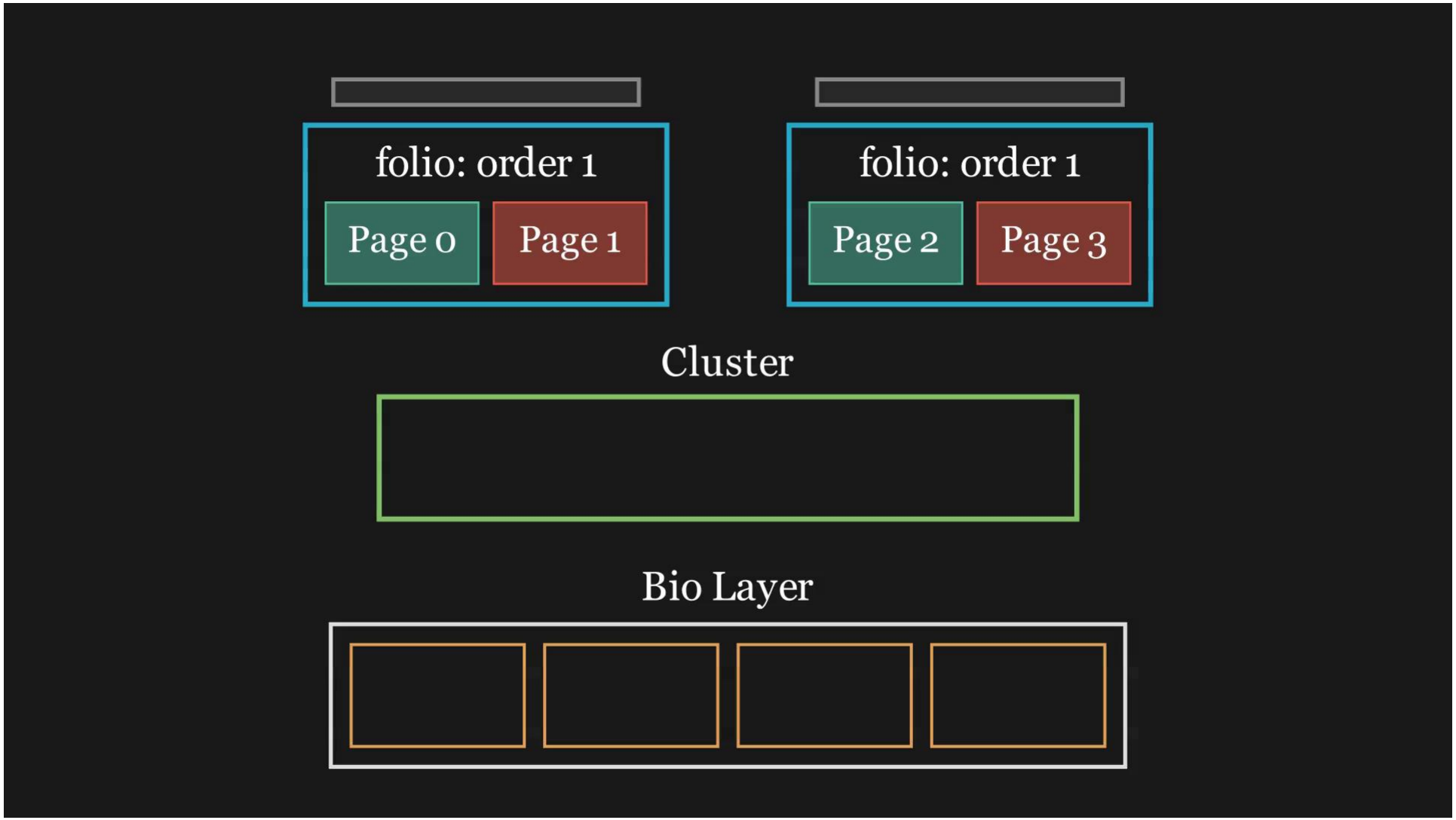
压缩文件folios 脏页回写：延迟解锁

- 原F2FS解锁策略：
  - 压缩文件脏页写回时高阶folio同样可跨越多个压缩/非压缩簇。高阶folio需要其所有子部分都被提交写io以后才可解锁。然而F2FS现有实现为了防止脏页写回和缓存写的并发死锁问题,会在folio每次写非压缩簇时将整个簇中的folio全部解锁,再重新上锁。这将使得跨簇高阶folio陷入”解锁,上锁,解锁,上锁 ……”的循环。锁的保护意义失效。
- 改进方案：
  - 因为folio总是先进压缩上下文,下次迭代后才能提交io,因而在脏区间查找循环结束后,我们打上延迟解锁标志位。
  - 当压缩 /非压缩簇写回逻辑识别到folio具备延迟解锁标志位,此时才将folio解锁。

```
enum {
    PAGE_PRIVATE_NOT_POINTER,
    PAGE_PRIVATE_ONGOING_MIGRATION
    PAGE_PRIVATE_INLINE_INODE,
    PAGE_PRIVATE_REF_RESOURCE,
    PAGE_PRIVATE_ATOMIC_WRITE,
    PAGE_PRIVATE_DEFERRED_UNLOCK,
    PAGE_PRIVATE_MAX
};
```

```
/* in f2fs_write_raw_pages / f2fs_write_compressed_pages */
if(folio_order(folio)>0&&fifs){
    if(f2fs_folio_private_deferred_unlock(folio)) {
        f2fs_clear_folio_private_deferred_unlock(folio);
        folio_unlock(folio);
    }
}
```

压缩文件folios 脏页回写：延迟解锁



其他工作：

- 普通文件iomap buffered read处理文件空洞的时候,同时有分配了dnode的NULL_ADDR和未分配dnode的空洞,为f2fs_map_blocks新增F2FS_MAP_NODNODE来区分两种不同的空洞处理逻辑。
- 在GC代码中使用f2fs_iomap_folio_state,GC产生脏页时只对folio内具体索引的子页标脏。
- 为f2fs_io_info增加idx和cnt成员,将其重解释成表示旧块首地址为old_blkaddr,新块首地址为new_blkaddr且对应的folio内(不是整个文件内的页偏移)首个页索引为idx,长度为cnt的一段连续块区间。
- 为inode增加i_iomap_seq成员,用于普通文件iomap buffered write中iomap_begin中在iomap中缓存的块映射信息被并发改动的问题。
- 在自己.put_folio钩子函数加上inode_inc_dirty_pages_multiple,防止iomap_buffered_write时进入balance_dirty_pages时发现f2fs_inode中脏页计数不增加导致一直跳过回写造成buffered write任务卡死。

.....

压缩文件冷读性能提升

```
文件: /mnt/f2fs/com_512M.txt
偏移: 0 字节 (0)
大小: 536870912 字节 (512M)
------------------
正在执行 dd 读取...
536870912 bytes (537 MB, 512 MiB) copied, 1 s, 377 MB/s
1+0 records in
1+0 records out
536870912 bytes (537 MB, 512 MiB) copied, 1.42991 s, 375 MB/s
读取操作完成。
==> 清理: 根据选项或文件状态，无需删除。
```

```
文件: /mnt/f2fs/com_512M.txt
偏移: 0 字节 (0)
大小: 536870912 字节 (512M)
------------------
正在执行 dd 读取...
536870912 bytes (537 MB, 512 MiB) copied, 1 s, 536 MB/s
1+0 records in
1+0 records out
536870912 bytes (537 MB, 512 MiB) copied, 1.00454 s, 534 MB/s
读取操作完成。
==> 清理: 根据选项或文件状态，无需删除。
```

- 虚拟机配置:
  - 内存:8G
  - 页面大小:4KB
  - PCIe:4.0

压缩文件读性能提升1.4至1.5倍

压缩文件冷写性能提升

```
ash ./rw_test.sh -k -f /mnt/f2fs/com_512M.txt w 0 512Mk -f /mnt/f2fs/com_512M.
--- 写入操作 ---
文件: /mnt/f2fs/com_512M.txt
偏移: 0 字节 (0)
大小: 536870912 字节 (512M)
------------------
正在执行 dd 写入 (数据源: /dev/zero)...
536870912 bytes (537 MB, 512 MiB) copied, 1 s, 511 MB/s
1+0 records in
1+0 records out
536870912 bytes (537 MB, 512 MiB) copied, 1.05311 s, 510 MB/s
写入操作完成。
==> 清理: 根据选项或文件状态，无需删除。
```

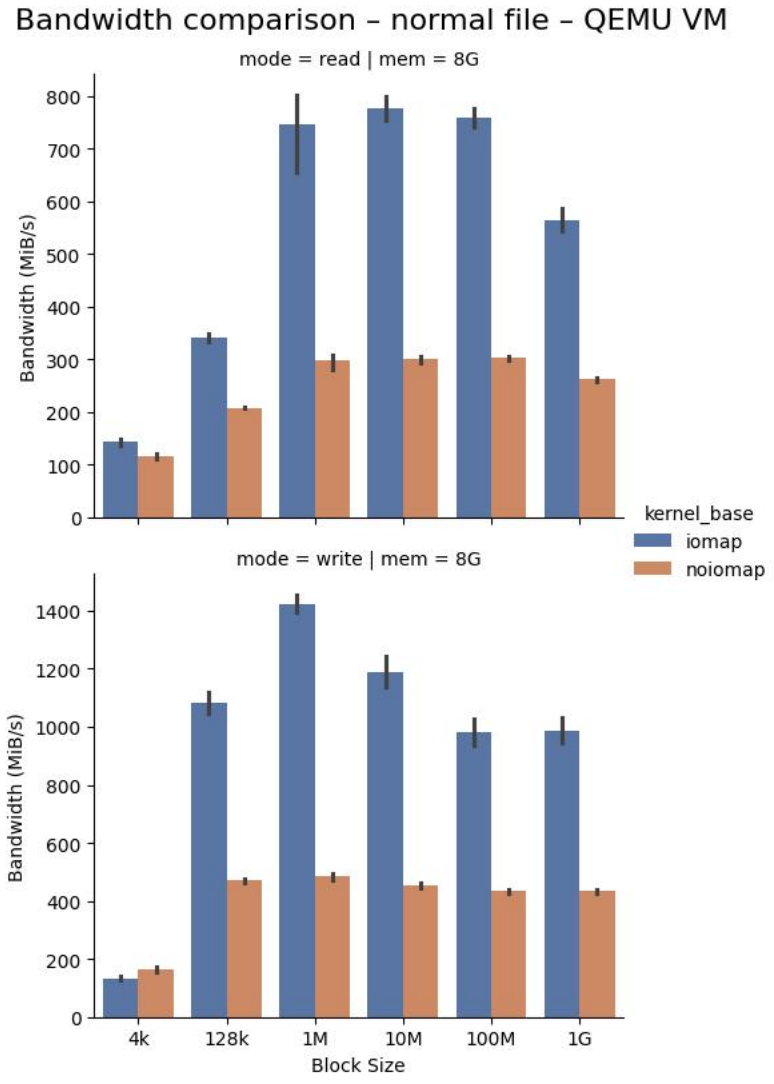
```
ash ./rw_test.sh -k -f /mnt/f2fs/com_512M.txt w 0 512Mk -f /mnt/f2fs/com_512M.txt w
--- 写入操作 ---
文件: /mnt/f2fs/com_512M.txt
偏移: 0 字节 (0)
大小: 536870912 字节 (512M)
------------------
正在执行 dd 写入 (数据源: /dev/zero)...
1+0 records in
1+0 records out
536870912 bytes (537 MB, 512 MiB) copied, 0.500432 s, 1.1 GB/s
写入操作完成。
==> 清理: 根据选项或文件状态，无需删除。
```

- 虚拟机配置:
  - 内存:8G
  - 页面大小:4KB
  - PCIe:4.0

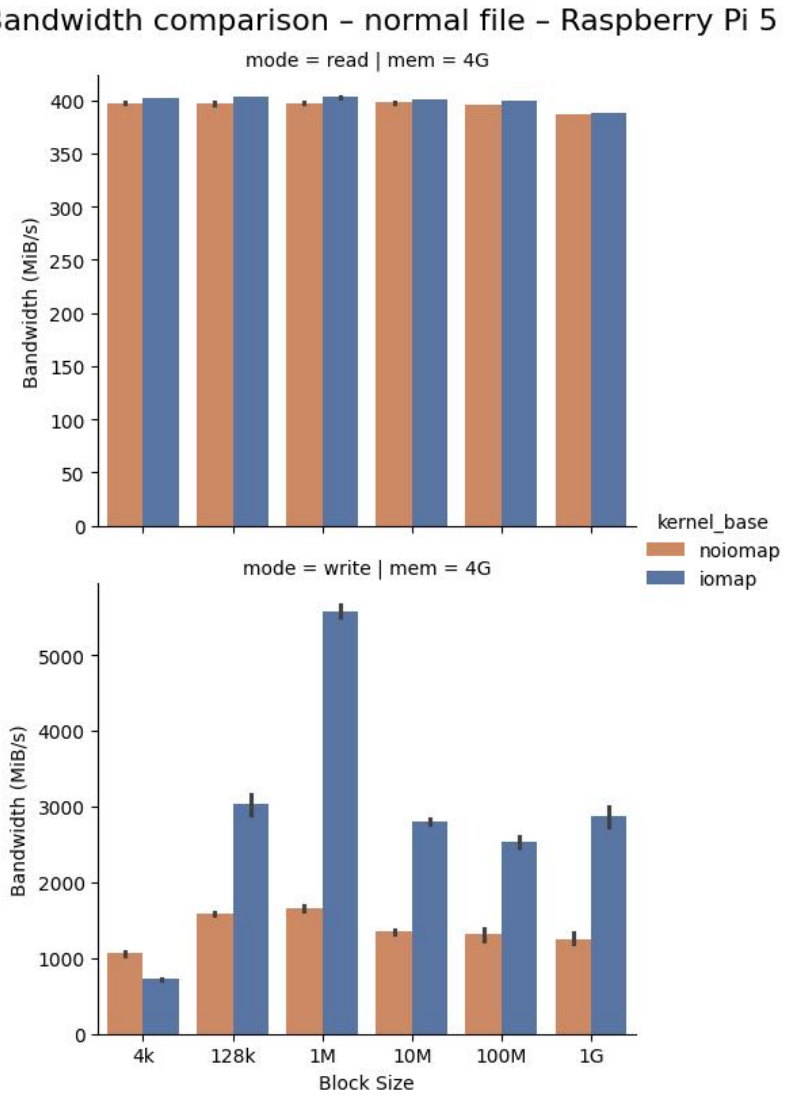
压缩文件冷buffered write性能提升2倍

非压缩文件性能benchmark

非压缩文件缓存读写测试带宽(虚拟机)



非压缩文件缓存读写测试带宽(树莓派)



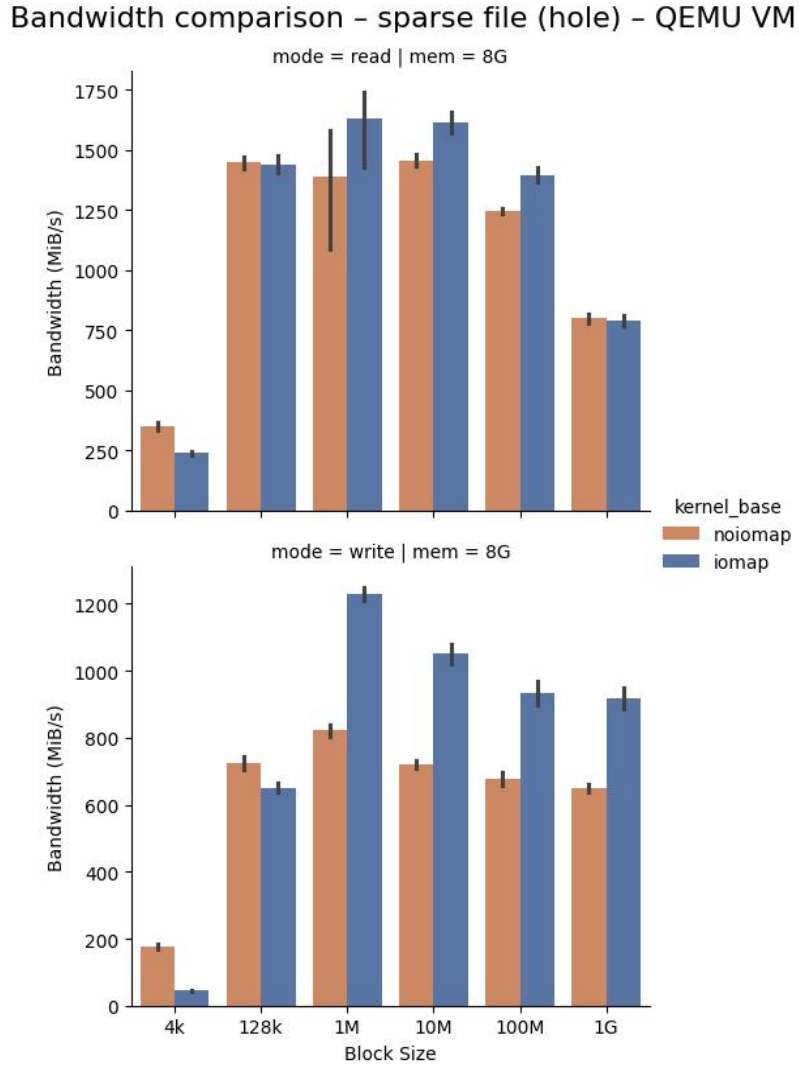
- 虚拟机配置:
  - 内存:8G
  - 页面大小:4KB
  - PCIe:3.0
- 树莓派配置:
  - 内存:4G
  - 页面大小:16KB
  - PCIe:2.0

在虚拟机上非压缩文件缓存读性能最高提升2倍,缓存写性能最高提升2.5倍!

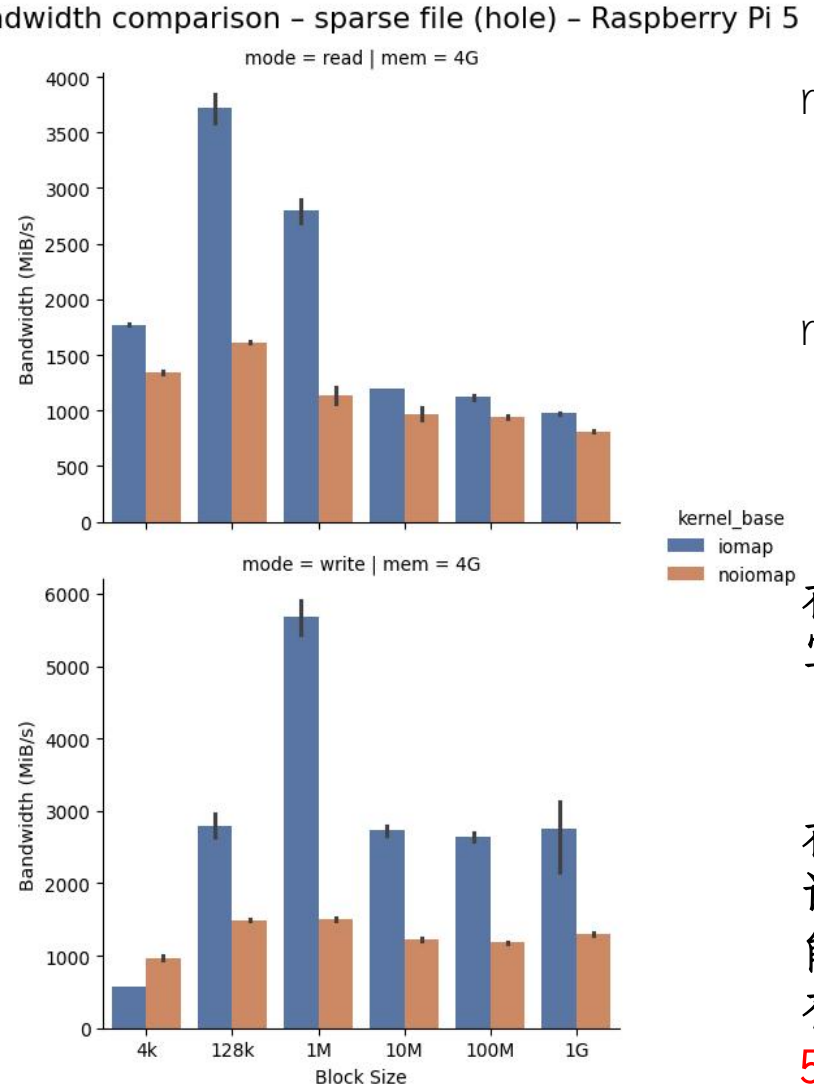
在树莓派上非压缩文件缓,缓存写性能更是最高提升至将近3倍,内存写入速率最高可达5500MB/s!

稀疏文件性能benchmark

稀疏文件缓存读写测试带宽(虚拟机)



稀疏文件缓存读写测试带宽(树莓派)



- 虚拟机配置:
  - 内存:8G
  - 页面大小:4KB
  - PCIe:3.0
- 树莓派配置:
  - 内存:4G
  - 页面大小:16KB
  - PCIe:2.0

在虚拟机上稀疏文件缓存写性能最高提升1.5倍!

在树莓派上稀疏文件缓存读性能提升两倍,缓存写性能最高提升至将近4倍,内存写入速率最高可达近5800MB/s!

RFC Patch提交

[f2fs-dev] [RESEND RFC PATCH 0/9] f2fs: Enable buffered read/write large folios support with extended iomap
 2025-08-13 9:37 UTC (10+ messages)
` [f2fs-dev] [RFC PATCH 1/9] f2fs: Introduce f2fs_iomap_folio_state
` [f2fs-dev] [RFC PATCH 2/9] f2fs: Integrate f2fs_iomap_folio_state into f2fs page private helpers
` [f2fs-dev] [RFC PATCH 3/9] f2fs: Using `folio_detach_f2fs_private` in invalidate and release folio
` [f2fs-dev] [RFC PATCH 4/9] f2fs: Convert outplace write path page private funcions to folio private functions
` [f2fs-dev] [RFC PATCH 5/9] f2fs:Refactor `f2fs_is_compressed_page` to `f2fs_is_compressed_folio`
` [f2fs-dev] [RFC PATCH 6/9] f2fs: Extend f2fs_io_info to support sub-folio ranges
` [f2fs-dev] [RFC PATCH 7/9] f2fs:Make GC aware of large folios
` [f2fs-dev] [RFC PATCH 8/9] f2fs: Introduce F2FS_GET_BLOCK_IOMAP and map_blocks he lpers
` [f2fs-dev] [RFC PATCH 9/9] f2fs: Enable buffered read/write path large folios support for normal and atomic file with iomap

linux-f2fs-devel.lists.sourceforge.net archive mirror

search help / color / mirror / Atom feed

* **[f2fs-dev] [RESEND RFC PATCH 0/9] f2fs: Enable buffered read/write large folios support with extended iomap**
@ 2025-08-13 9:37 Nanzhe Zhao
  2025-08-13 9:37 ` [f2fs-dev] [RFC PATCH 1/9] f2fs: Introduce f2fs_iomap_folio_state Nanzhe Zhao
                  ` (8 more replies)
  0 siblings, 9 replies; 10+ messages in thread
From: Nanzhe Zhao @ 2025-08-13 9:37 UTC (permalink / raw)
  To: Jaegeuk Kim, linux-f2fs-devel, linux-fsdevel
  Cc: Nanzhe Zhao, Barry Song, Matthew Wilcox, Yi Zhang

[-- Warning: decoded text below may be mangled, UTF-8 assumed --]
[-- Attachment #1: Type: text/plain; charset=y, Size: 6551 bytes --]

Resend: original patch was misspelling
the linux-f2fs-devel@lists.sourceforge.net address.
No code changes.

This RFC series enable buffered read/write paths large folio support
with F2FS-specific extended iomap, combined with some other preparation
work for large folio integration.

汇报提纲

1. 社区进展回顾
2. 问题背景
3. 项目功能解析
4. **未来进展规划**

未来进展规划

- 使用large folios使F2FS可以兼容块大小 > 页大小的场景
- 尝试将压缩文件buffered read和page writeback集成到原生iomap 框架.....
- 尽可能消除0阶folio分配f2fs_iomap_folio_state的内存开销.....或想出比f2fs_iomap_folio_state更好的方案.....
- 让GC分配large folios.....
- 让seek,fallocate,pgmkwrite,fiemap支持iomap.....
- 进行大量测试,包括白盒测试,用故障注入手段等,尽可能使得代码稳定性提升。
- 发现更多iomap框架提供而F2FS需要拓展的点.....
- 压缩文件large folios配合F2FS动态大小簇等新压缩技术,真正提高压缩率

南京工業大學
NANJING TECH UNIVERSITY

CHINA LINUX KERNEL
中国Linux内核开发者大会

- 我的联系方式:
  - 手机号:18921179826
  - 微信号:nzzhao_zstz
  - 邮箱:nzzhao@126.com

南京工業大學
NANJING TECH UNIVERSITY

CHINA LINUX KERNEL
中国Linux内核开发者大会

特别感谢:万夕里老师,宋宝华老师,俞超老师,张翼老师

THANKS