CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

Parallelizing filesystem writeback

王誉霏　vivo 存储系统工程师
张细锐　vivo 存储系统工程师

CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

目录

CONTENTS

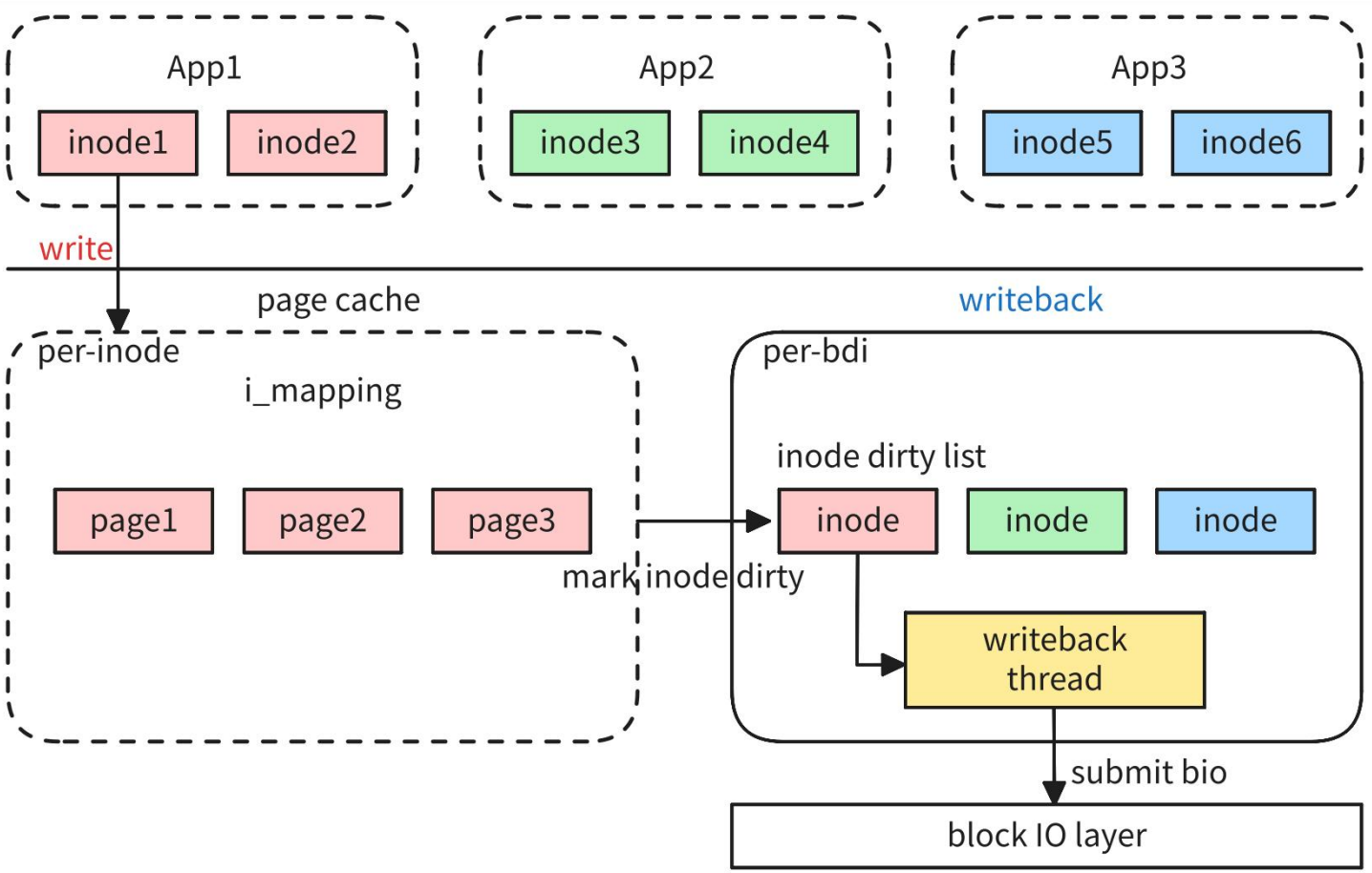
Part One

Writeback背景

Writeback背景

buffered IO 机制

- buffered I/O 通过 Page Cache 实现，由write（写入缓冲合并）和writeback（异步延迟回写）两部分组成
- 性能瓶颈：高负载情况下，应用程序多文件并发写入时，脏页回写效率低，导致page cache中脏页积压及应用程序阻塞



Writeback背景

writeback机制

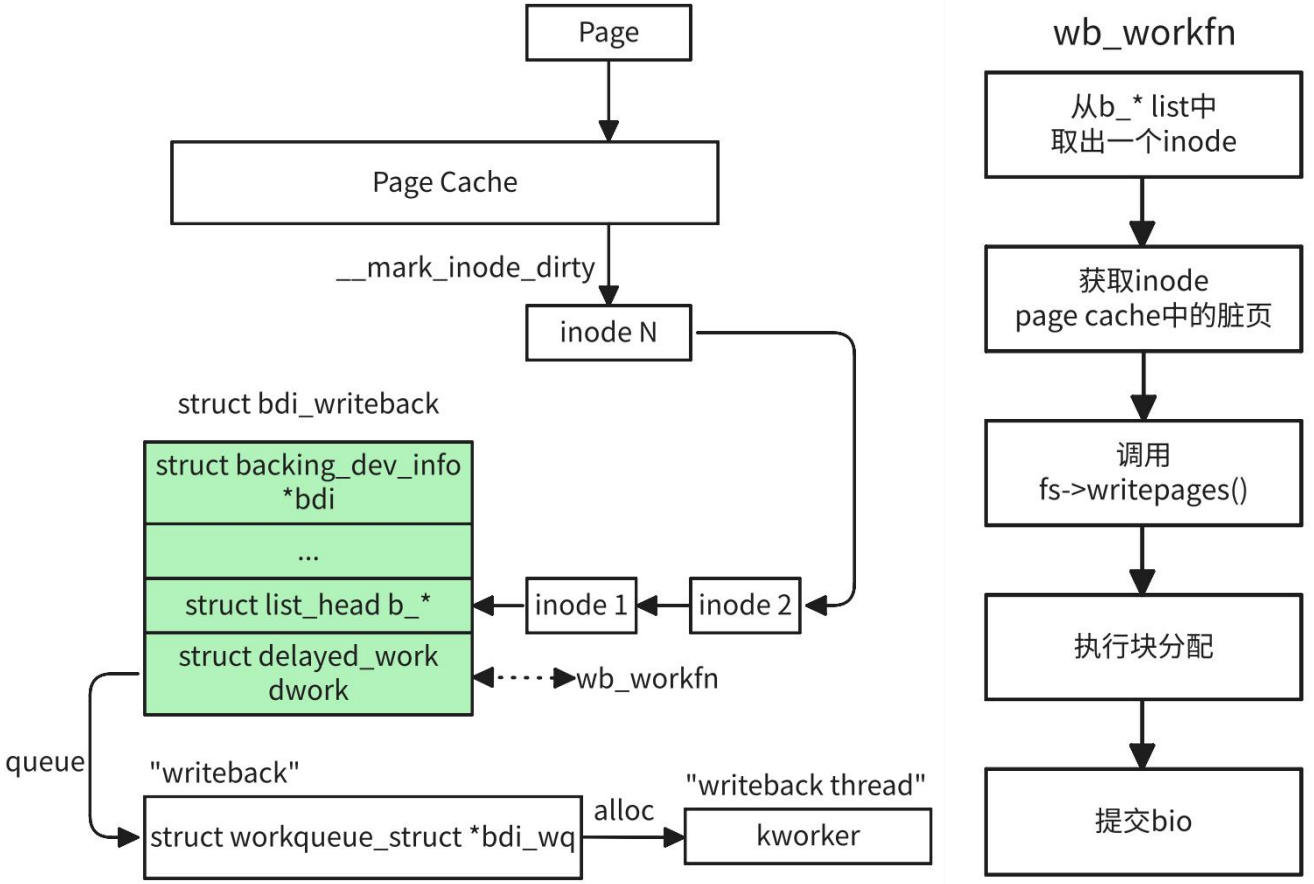
- Backing device info (BDI, struct backing_dev_info): 核心writeback结构体，作为page cache和block IO层间的桥梁
- Bdi writeback (struct bdi_writeback): 存储需要回写的数据(b_* dirty list)；触发相应的回写线程(dwork)
- 映射关系：block device → backing_dev_info → bdi_writeback → single writeback thread

```
struct backing_dev_info {
    ...
    struct bdi_writeback wb;  /* the root writeback info for this bdi */
    struct list_head wb_list; /* list of all wbs */
#ifdef CONFIG_CGROUP_WRITEBACK
    struct radix_tree_root cgwb_tree; /* radix tree of active cgroup wbs */
    struct mutex cgwb_release_mutex;  /* protect shutdown of wb structs */
    struct rw_semaphore wb_switch_rwsem; /* no cgwb switch while syncing */
#endif
    wait_queue_head_t wb_waitq;
    ...
}

struct bdi_writeback {
    struct backing_dev_info *bdi;   /* our parent bdi */

    unsigned long state;        /* Always use atomic bitops on this */
    unsigned long last_old_flush;   /* last old data flush */

    //inode链表
    struct list_head b_dirty;   /* dirty inodes */
    struct list_head b_io;      /* parked for writeback */
    struct list_head b_more_io; /* parked for more writeback */
    struct list_head b_dirty_time;  /* time stamps are dirty */
    spinlock_t list_lock;       /* protects the b_* lists */
    ...
    //用于触发writeback的延迟工作项
    struct delayed_work dwork;  /* work item used for writeback */
    ...
}
```



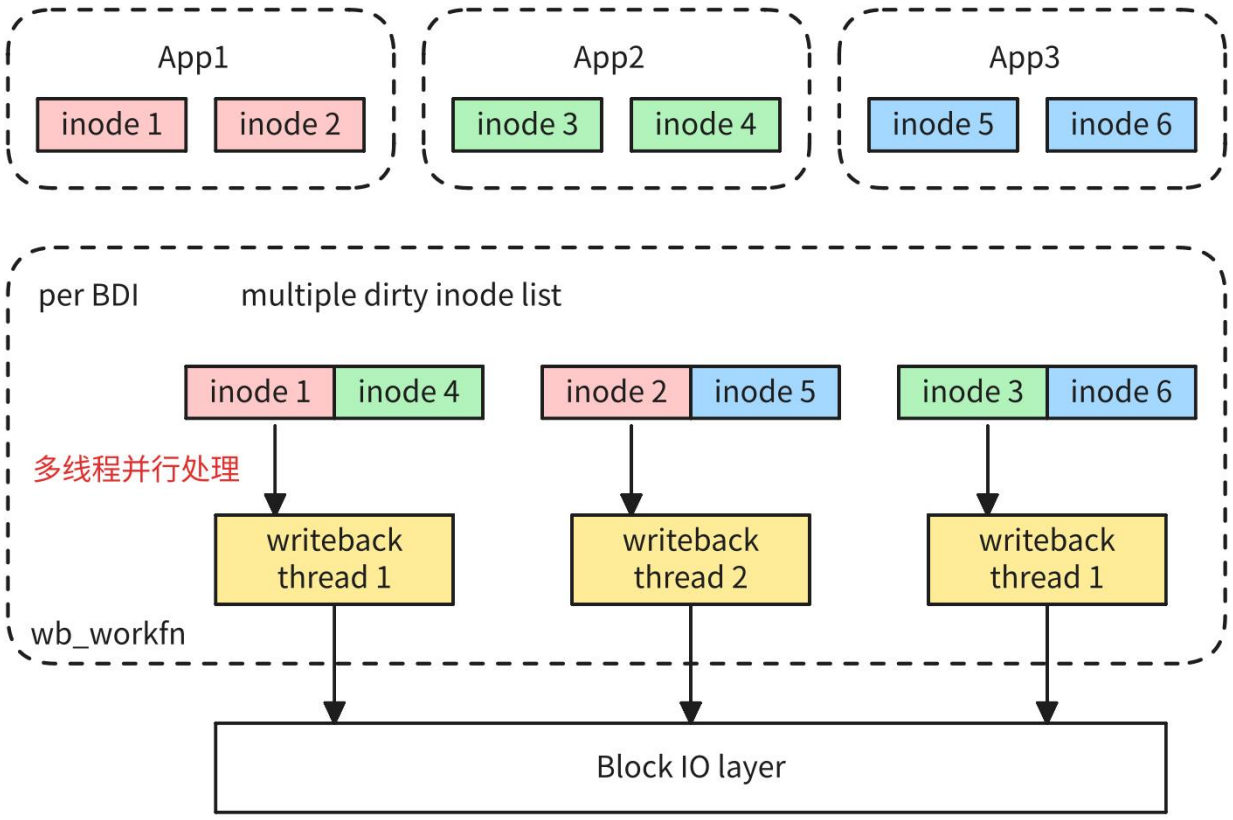
Part Two

Parallelizing writeback 探索与实践

Parallelizing writeback探索与实践

Parallelizing writeback实现

- 核心思想：允许文件系统为每个BDI创建多个writeback线程
- 实现：从struct backing_dev_info中抽象出struct bdi_writeback_ctx，同时管理多个bdi_writeback
- 默认配置对应CPU数量的bdi_writeback_ctx

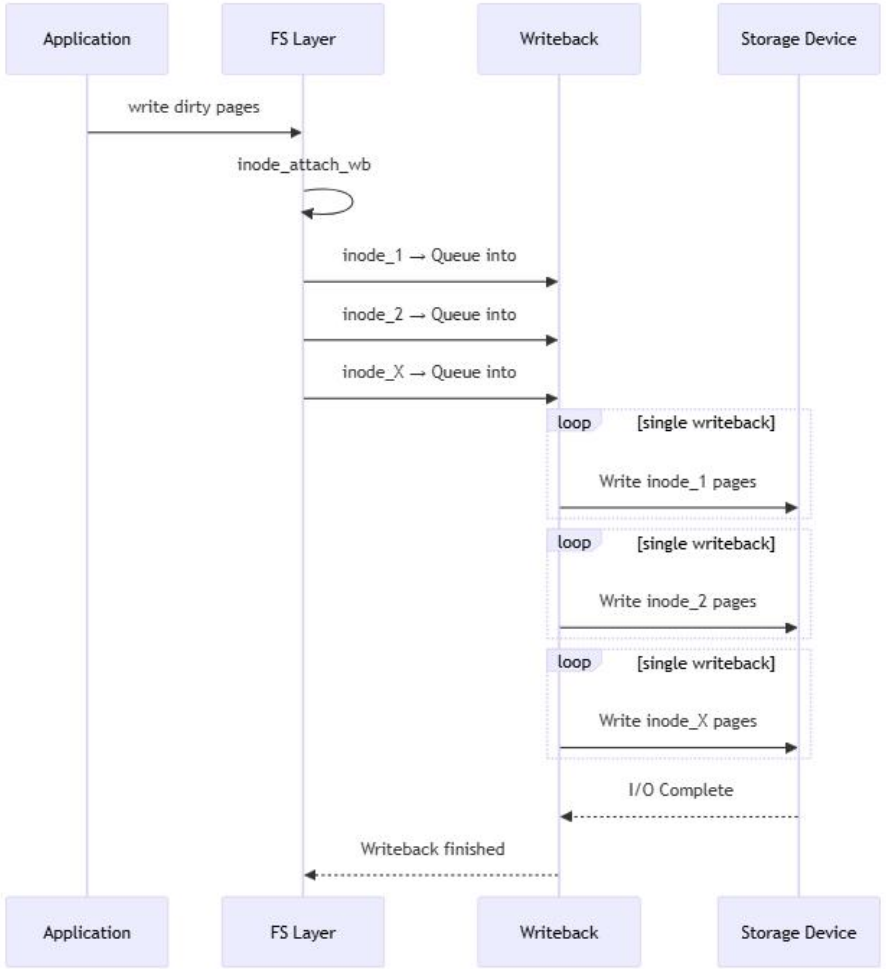


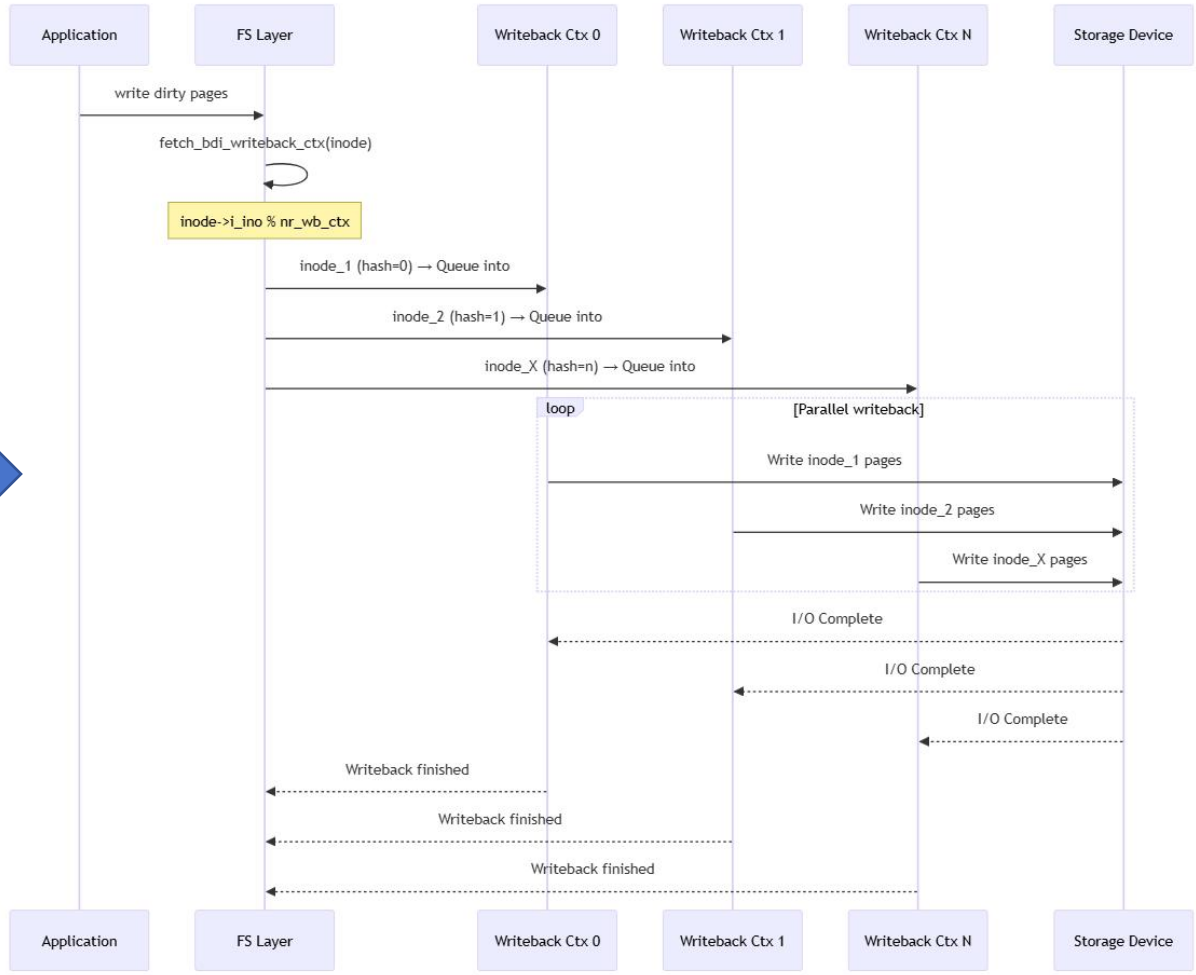

```
//before
struct backing_dev_info {
    ...
    struct bdi_writeback wb;  /* the root writeback info for this bdi */
    struct list_head wb_list; /* list of all wbs */
#ifdef CONFIG_CGROUP_WRITEBACK
    struct radix_tree_root cgwb_tree; /* radix tree of active cgroup wbs */
    struct mutex cgwb_release_mutex;  /* protect shutdown of wb structs */
    struct rw_semaphore wb_switch_rwsem; /* no cgwb switch while syncing */
#endif
    wait_queue_head_t wb_waitq;
    ...
}
//after
struct backing_dev_info {
    ...
    int nr_wb_ctx;
    struct bdi_writeback_ctx **wb_ctx_arr;
#ifdef CONFIG_CGROUP_WRITEBACK
    struct mutex cgwb_release_mutex;  /* protect shutdown of wb structs */
#endif
    ...
}
struct bdi_writeback_ctx {
    struct bdi_writeback wb;  /* the root writeback info for this bdi */
    struct list_head wb_list; /* list of all wbs */
#ifdef CONFIG_CGROUP_WRITEBACK
    struct radix_tree_root cgwb_tree; /* radix tree of active cgroup wbs */
    struct rw_semaphore wb_switch_rwsem; /* no cgwb switch while syncing */
#endif
    wait_queue_head_t wb_waitq;
}
```

Parallelizing writeback探索与实践

writeback流程对比

- Before: block device → backing_dev_info → single bdi_writeback → single writeback thread
- After: block device → backing_dev_info → multiple bdi_writeback → multiple writeback thread

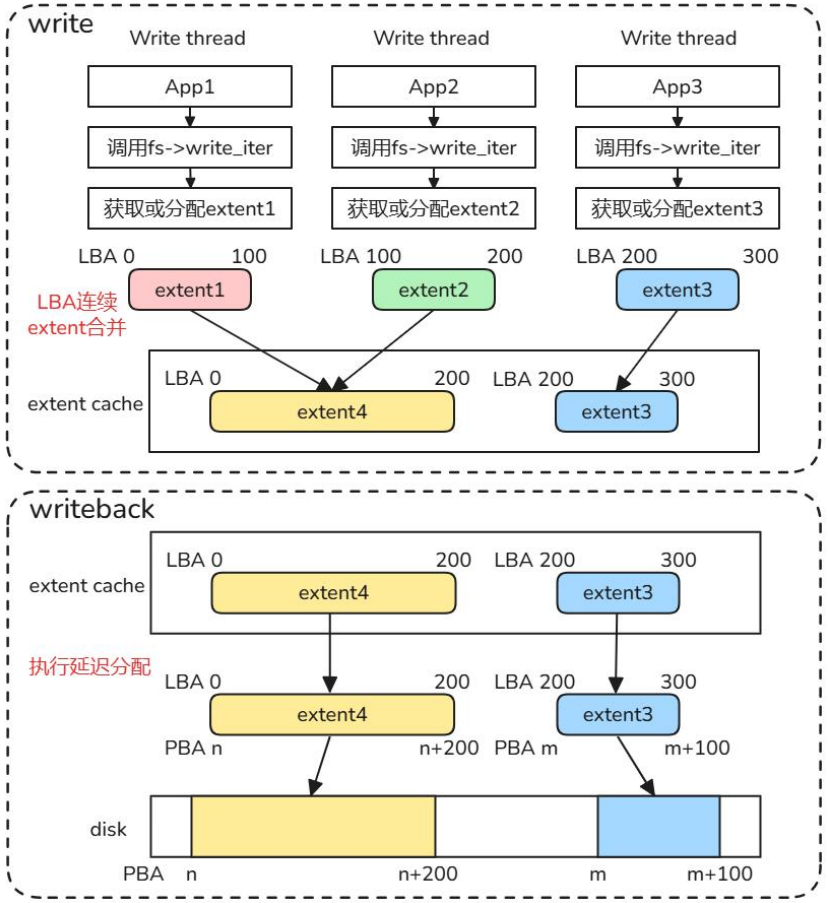


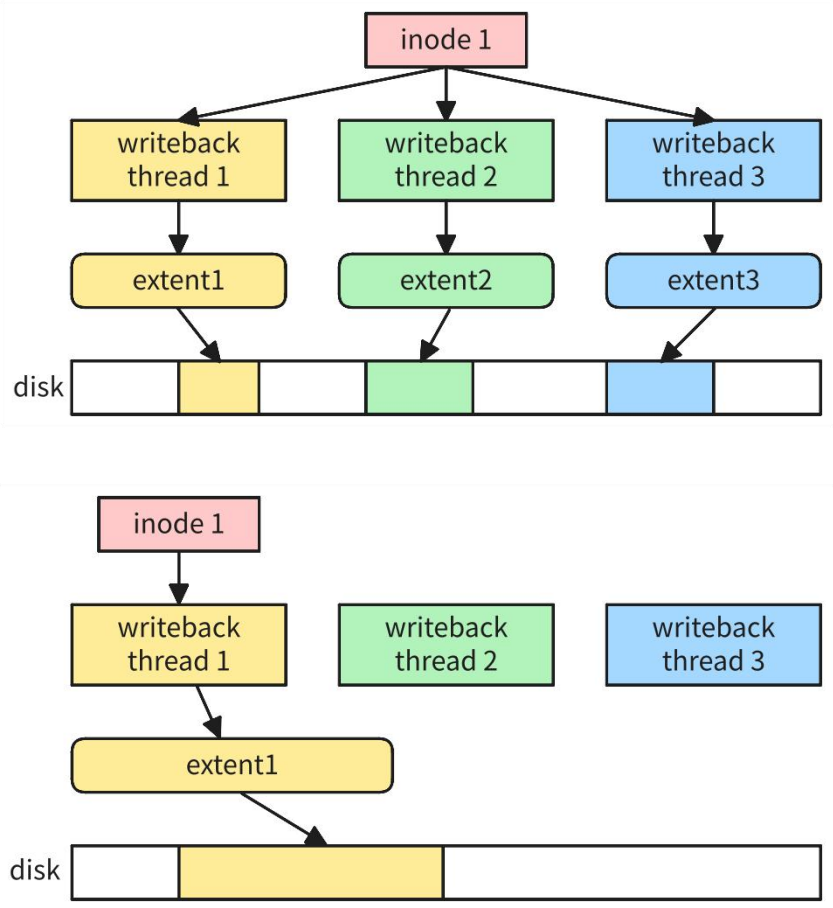


Parallelizing writeback探索与实践

writeback并行化带来的挑战 1 ——加剧文件系统碎片化

- 延迟分配：同一个inode的extent可以合并后分配物理地址
- 问题：如果多个线程同时回写一个inode中的不同页，则会阻碍延迟分配，导致碎片化问题
- 解决方案：优化inode分配策略，每个inode只关联到一个writeback_ctx，避免碎片化的产生

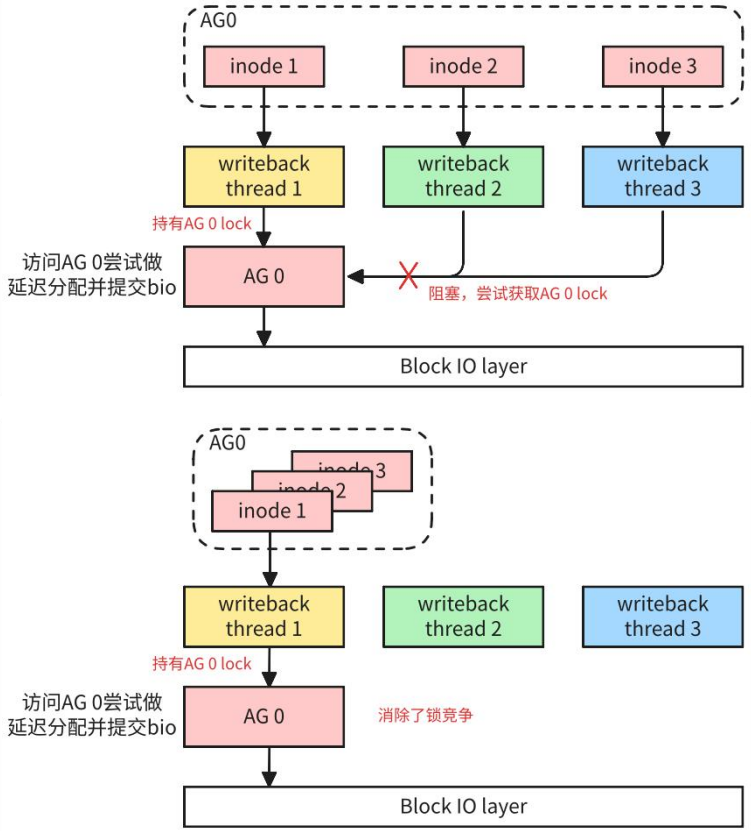




Parallelizing writeback探索与实践

writeback并行化带来的挑战 2 —— 与文件系统特性不匹配

- 以XFS为例：每个inode及其数据倾向于分配到同一个Allocation Group（AG)中，如果多个writeback线程同时回写一个AG中的inode，则有可能导致AG锁竞争
- 解决方案：设计一个文件系统接口，允许文件系统按照其特性来配置inode的分配策略；对于XFS，将对应AG的inode全部分配到同一writeback_ctx中
- 提交链接：https://lore.kernel.org/linux-fsdevel/20250914121109.36403-1-wangyufei@vivo.com/




```
@@ -144,7 +144,10 @@ static inline struct bdi_writeback_ctx *
 fetch_bdi_writeback_ctx(struct inode *inode)
 {
     struct backing_dev_info *bdi = inode_to_bdi(inode);
+    struct super_block *sb = inode->i_sb;

+    if (sb->s_op->get_inode_wb_ctx_idx)
+        return bdi->wb_ctx_arr[sb->s_op->get_inode_wb_ctx_idx(inode, bdi->nr_wb_ctx)];
     return bdi->wb_ctx_arr[inode->i_ino % bdi->nr_wb_ctx];
 }

@@ -2491,6 +2491,7 @@ struct super_operations {
      */
     int (*remove_bdev)(struct super_block *sb, struct block_device *bdev);
     void (*shutdown)(struct super_block *sb);
+    unsigned int (*get_inode_wb_ctx_idx)(struct inode *inode, int nr_wb_ctx);
 };

@@ -1295,6 +1308,7 @@ static const struct super_operations xfs_super_operations = {
     .free_cached_objects    = xfs_fs_free_cached_objects,
     .shutdown       = xfs_fs_shutdown,
     .show_stats     = xfs_fs_show_stats,
+    .get_inode_wb_ctx_idx   = xfs_fs_get_inode_wb_ctx_idx,
 };
```

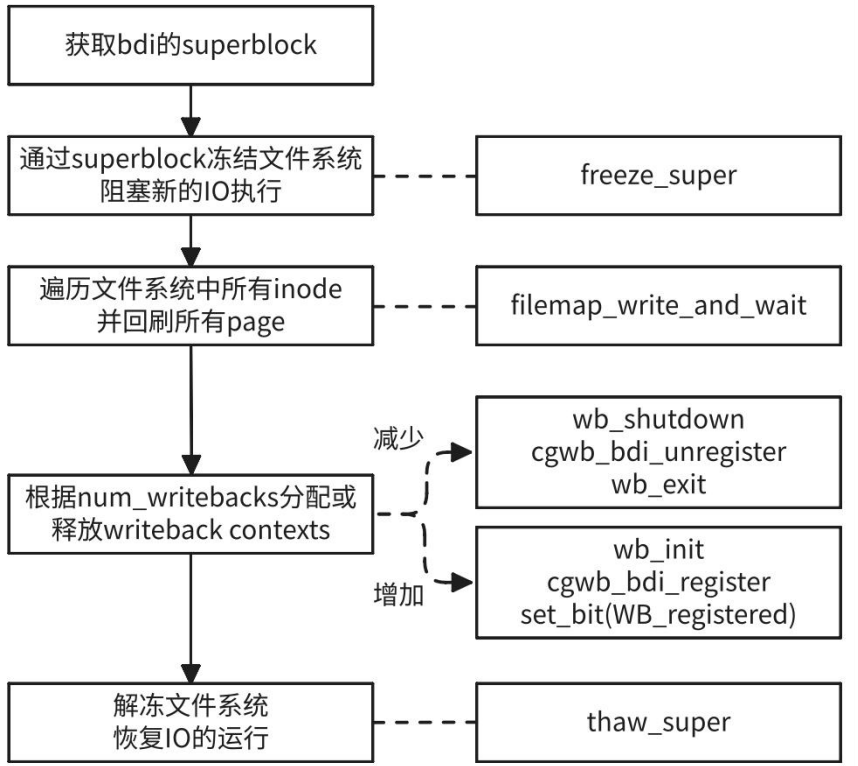
Parallelizing writeback探索与实践

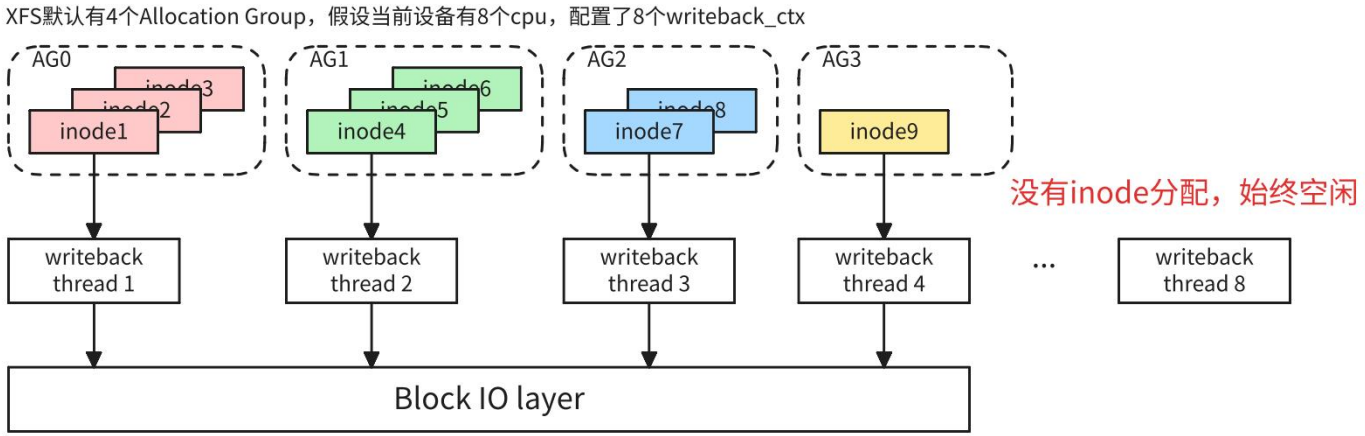
可动态配置的writeback特性

- 问题：writeback_ctx的数量默认配置为当前系统中CPU的数量，但在某些情况下，该配置不一定能获得最优性能
- 解决方案：设计一个sysfs节点“nwritebacks”用于调整writeback_ctx的数量；用户可通过如下指令进行调整：

```
echo <num_writebacks> > /sys/class/bdi/<major>:<minor>/nwritebacks
```

- 有助于针对不同类型的I/O负载和文件系统特性调整并行回写的行为
- 提交链接：https://lore.kernel.org/linux-fsdevel/20250825122931.13037-1-wangyufei@vivo.com/





Parallelizing writeback探索与实践

buffered IO测试

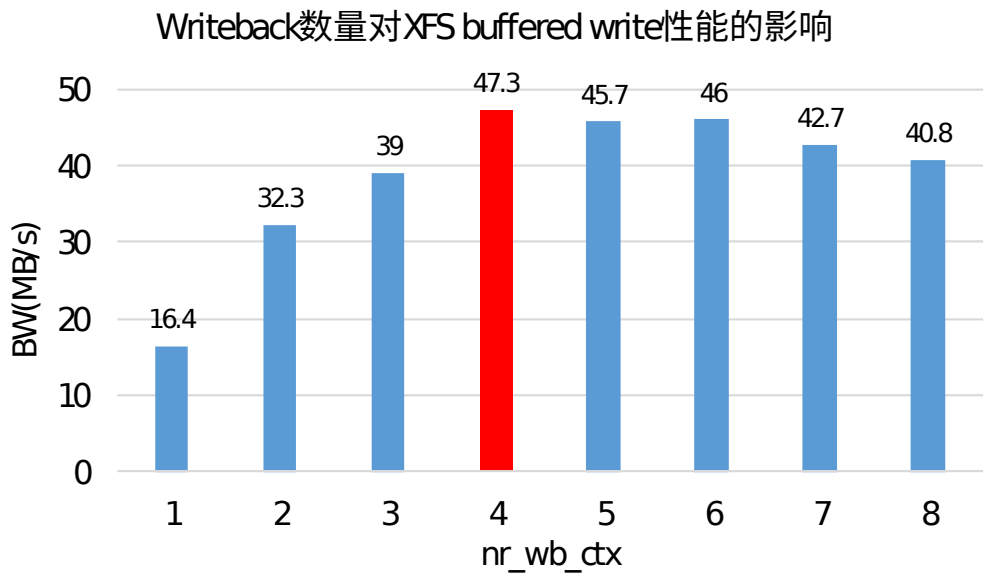
实机测试

- 测试结果：buffered-write性能提升22%
- 系统参数：文件系统：F2FS；CPUs：8；系统RAM：11G
- 测试命令： fio --directory=/data --name=test --bs=4k --iodepth=1024 --rw=randwrite --ioengine=io_uring --time_based=1 -runtime=60 --numjobs=8 --size=450M --direct=0

QEMU测试

- 测试结果：对于XFS，在writeback数量等于AG数量时，可以获得最优性能
- 系统参数：QEMU：模拟20GB NVMe SSD；文件系统：XFS；CPUs：8；系统RAM：4G

Configuration	single writeback	parallelizing writeback
IOPS(avg)	239k	293k
BW(MB/s)	978	1198(+22%)
Latency(usec avg)	34226	27779
Write Duration(msec)	60225	60455



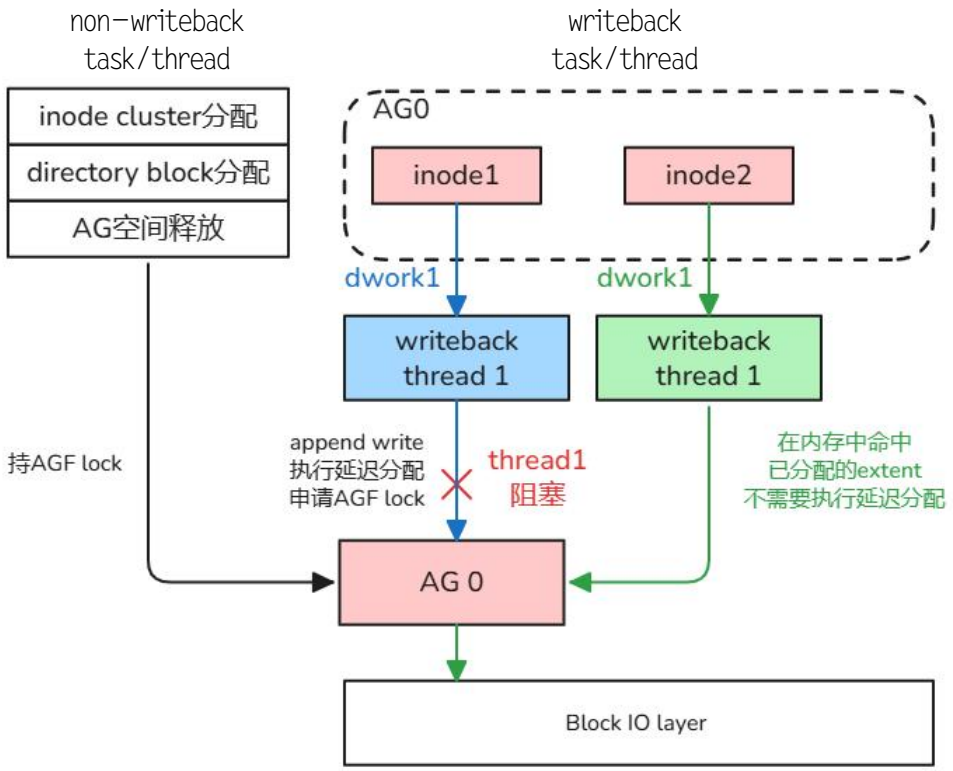
Part Three

未来展望

未来展望

bdi_writeback内部parallelizing writeback实现

- 以XFS为例：在系统实际运行的过程中，可能存在非writeback任务持有AG lock的情况，假设此时某个inode的回写涉及延迟分配，就会导致整个writeback线程阻塞，即便链表中的其他inode可能不需要延迟分配
- 实现：在bdi_writeback内分配多个dwork，用于触发多个writeback线程
- 当前存在的问题：数据完整性、数据写入顺序…




```
#define NUM_WB          4
struct mul_dwork {
    struct delayed_work dwork;
    struct bdi_writeback *p_wb;
};
struct bdi_writeback {
    struct backing_dev_info *bdi; /* our parent bdi */
    ...
    struct mul_dwork dwork[NUM_WB]; /* multiple dworks */
    int wb_idx;
    ...
}

static void wb_wakeup(struct bdi_writeback *wb)
{
    spin_lock_irq(&wb->work_lock);
    if (test_bit(WB_registered, &wb->state)) {
        mod_delayed_work(bdi_wq,
                &wb->wb_dwork[wb->wb_idx].dwork,
                0);
        wb->wb_idx = (wb->wb_idx + 1) % NUM_WB;
    }
    spin_unlock_irq(&wb->work_lock);
}
```

未来展望

文件碎片化

- 在实际使用和测试的过程中发现，文件碎片化问题依旧存在
- 测试方法

```
//1
fio --directory=/data --name=test --bs=4k --iodepth=1024 --rw=randwrite --
ioengine=io_uring --time_based=1 -runtime=60 --numjobs=8 --size=450M --direct=0
//2
filefrag /mnt/fs/testfile
```

- 测试结果

Configuration	Filesystem	# of extents (avg)	
		single writeback	parallelizing writeback
实机	F2FS	61533	67590
QEMU	F2FS	69068	71973
QEMU	XFS	66114	69518

THANKS