cgroup v2 block io note

latency方式io控制器的实测可用条件：

- ext4
  - read ok
  - write ok
- f2fs
  - read ok (PS: 注意设置正确的blk device, 注意读不同的文件)
  - write ok
  - PS: rq_qos似乎会降低磁盘吞吐量？
  - TODO: **需要设计一个测试，记录request层每秒的吞吐量，证实有没有降低？**

代码调用栈

readahead 调用栈 (from page_fault)

```
/****************************************************************/ //CX____
__do_page_cache_readahead[157]
//CPU: 0 PID: 128 Comm: mdev Not tainted 4.19.100+ #14
//Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.10.2-1ubuntu1 04/01/2014
//Call Trace:

dump_stack+0x71/0x97
__do_page_cache_readahead+0xbd/0x1c3
? find_get_entry+0x1e/0x176
? pagecache_get_page+0x2d/0x2e8
filemap_fault+0x255/0x661
ext4_filemap_fault+0x31/0x44 __do_fault+0x34/0xe0 __handle_mm_fault+0x106b/0x1535
handle_mm_fault+0xe0/0x24e
__do_page_fault+0x3eb/0x57d
do_page_fault+0x30/0xe8
? page_fault+0x8/0x30
page_fault+0x1e/0x30
```

readahead 调用栈 (from syscall)

```
/****************************************************************/ //CX____
__do_page_cache_readahead[157] //CPU: 0 PID: 125 Comm: init Not tainted 4.19.100+ #14
```

//Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.10.2-1ubuntu1 04/01/2014
//Call Trace:

dump_stack+0x71/0x97 __do_page_cache_readahead+0xbd/0x1c3 ? d_absolute_path+0x6b/0x9c ondemand_readahead+0x1b5/0x2be page_cache_sync_readahead+0xaf/0xd4 generic_file_read_iter+0x305/0xa84 ext4_file_read_iter+0x53/0xdf __vfs_read+0x13f/0x16f vfs_read+0x91/0x13d kernel_read+0x31/0x42 prepare_binprm+0xfa/0x1cf __do_execve_file+0x52a/0x818 do_execve+0x2d/0x2f __x64_sys_execve+0x2b/0x32 do_syscall_64+0x5c/0x128 entry_SYSCALL_64_after_hwframe+0x44/0xa9

blk_cgroup_congest 调用栈

/**/ dump_stack+0x71/0x97
blk_cgroup_congested mem_cgroup_throttle_swaprate+0x1d/0x160
mem_cgroup_try_charge_delay+0x37/0x43
__handle_mm_fault+0xad1/0x1535
handle_mm_fault+0xe0/0x24e
__do_page_fault+0x3eb/0x57d
do_page_fault+0x30/0xe8
? page_fault+0x8/0x30
page_fault+0x1e/0x30

mem_cgroup_throttle_swaprate 调用栈

/**/ dump_stack+0x71/0x97 mem_cgroup_throttle_swaprate+0x1d/0x160 mem_cgroup_try_charge_delay+0x37/0x43 wp_page_copy+0x147/0x5bb do_wp_page+0x347/0x50e __handle_mm_fault+0x14d4/0x1535 ? __switch_to_asm+0x35/0x70 handle_mm_fault+0xe0/0x24e __do_page_fault+0x3eb/0x57d do_page_fault+0x30/0xe8 ? page_fault+0x8/0x30 page_fault+0x1e/0x30

iolatency done bio 调用栈

/**/ iolatency_check_latencies blkcg_iolatency_done_bio rq_qos_done_bio bio_endio req_bio_endio blk_update_request scsi_end_request scsi_io_completion scsi_finish_command scsi_softirq_done blk_done_softirq __do_softirq invoke_softirq irq_exit exiting_irq do_IRQ common_interrupt

vfs_read blkcg_iolatency_throttle 调用栈

/**/ __blkcg_iolatency_throttle blkcg_iolatency_throttle rq_qos_throttle blk_queue_bio generic_make_request submit_bio ext4_mpage_readpages ext4_readpages read_pages __do_page_cache_readahead ra_submit ondemand_readahead page_cache_async_readahead generic_file_buffered_read generic_file_read_iter ext4_file_read_iter call_read_iter new_sync_read __vfs_read vfs_read

vfs_read to lkcg_iolatency_throttle 调用栈

/**/ blkcg_iolatency_throttle(struct rq_qos * rqos, struct bio * bio, spinlock_t * lock) (/home/panard/linux-4.19.100/block/blk-iolatency.c:436) rq_qos_throttle(struct request_queue * q, struct bio * bio, spinlock_t * lock) (/home/panard/linux-4.19.100/block/blk-rq-qos.c:77) blk_queue_bio(struct request_queue * q, struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2060) generic_make_request(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2464) submit_bio(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2573) ext4_mpage_readpages(struct address_space * mapping, struct list_head * pages, struct page * page, unsigned int nr_pages, bool is_readahead) (/home/panard/linux-4.19.100/fs/ext4/readpage.c:293) ext4_readpages(struct file * file, struct address_space * mapping, struct list_head * pages, unsigned int nr_pages) (/home/panard/linux-4.19.100/fs/ext4/inode.c:3364) read_pages(struct address_space * mapping, struct file * filp, struct list_head * pages, unsigned int nr_pages, gfp_t gfp) (/home/panard/linux-4.19.100/mm/readahead.c:123) __do_page_cache_readahead(struct address_space * mapping, struct file * filp, unsigned long offset, unsigned long nr_to_read, unsigned long lookahead_size) (/home/panard/linux-4.19.100/mm/readahead.c:215) ra_submit() (/home/panard/linux-4.19.100/mm/internal.h:66) do_sync_mmap_readahead() (/home/panard/linux-4.19.100/mm/filemap.c:2467) filemap_fault(struct vm_fault * vmf) (/home/panard/linux-4.19.100/mm/filemap.c:2543) ext4_filemap_fault(struct vm_fault * vmf) (/home/panard/linux-4.19.100/fs/ext4/inode.c:6353) __do_fault(struct vm_fault * vmf) (/home/panard/linux-4.19.100/mm/memory.c:3269) do_read_fault() (/home/panard/linux-4.19.100/mm/memory.c:3681) do_fault() (/home/panard/linux-4.19.100/mm/memory.c:3810) handle_pte_fault() (/home/panard/linux-4.19.100/mm/memory.c:4041) __handle_mm_fault(struct vm_area_struct * vma, unsigned long address, unsigned int flags) (/home/panard/linux-4.19.100/mm/memory.c:4165) handle_mm_fault(struct vm_area_struct * vma, unsigned long address, unsigned int flags) (/home/panard/linux-4.19.100/mm/memory.c:4202) __do_page_fault(struct pt_regs * regs, unsigned long error_code, unsigned long address) (/home/panard/linux-4.19.100/arch/x86/mm/fault.c:1390) do_page_fault(struct pt_regs * regs, unsigned long error_code) (/home/panard/linux-4.19.100/arch/x86/mm/fault.c:1465) page_fault() (/home/panard/linux-4.19.100/arch/x86/entry/entry_64.S:1204) [Unknown/Just-In-Time compiled code] (Unknown Source:0) irq_stack_union (Unknown Source:0) [Unknown/Just-In-Time compiled code] (Unknown Source:0)

f2fs: (cgroup io部分与f2fs的实现有关联)

vfs_read to blkcg_iolatency_throttle 调用栈

```
/***********************************************************************************
blkcg_iolatency_throttle(struct rq_qos * rqos, struct bio * bio, spinlock_t * lock) (/
rq_qos_throttle(struct request_queue * q, struct bio * bio, spinlock_t * lock) (/home/
blk_queue_bio(struct request_queue * q, struct bio * bio) (/home/panard/linux-4.19.100
generic_make_request(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2
submit_bio(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2573)
__submit_bio() (/home/panard/linux-4.19.100/fs/f2fs/data.c:307)
f2fs_mpage_readpages(struct address_space * mapping, struct list_head * pages, struct
f2fs_read_data_pages(struct file * file, struct address_space * mapping, struct list_h
read_pages(struct address_space * mapping, struct file * filp, struct list_head * page
__do_page_cache_readahead(struct address_space * mapping, struct file * filp, unsigned
ra_submit() (/home/panard/linux-4.19.100/mm/internal.h:66)
do_sync_mmap_readahead() (/home/panard/linux-4.19.100/mm/filemap.c:2467)
filemap_fault(struct vm_fault * vmf) (/home/panard/linux-4.19.100/mm/filemap.c:2543)
f2fs_filemap_fault(struct vm_fault * vmf) (/home/panard/linux-4.19.100/fs/f2fs/file.c:
__do_fault(struct vm_fault * vmf) (/home/panard/linux-4.19.100/mm/memory.c:3269)
do_read_fault() (/home/panard/linux-4.19.100/mm/memory.c:3681)
do_fault() (/home/panard/linux-4.19.100/mm/memory.c:3810)
handle_pte_fault() (/home/panard/linux-4.19.100/mm/memory.c:4041)
__handle_mm_fault(struct vm_area_struct * vma, unsigned long address, unsigned int fla
handle_mm_fault(struct vm_area_struct * vma, unsigned long address, unsigned int flags
__do_page_fault(struct pt_regs * regs, unsigned long error_code, unsigned long address
do_page_fault(struct pt_regs * regs, unsigned long error_code) (/home/panard/linux-4.1
page_fault() (/home/panard/linux-4.19.100/arch/x86/entry/entry_64.S:1204)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
irq_stack_union (Unknown Source:0)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

f2fs read流程 到 generic_make_request 调用栈

```
/***********************************************************************************
generic_make_request(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2
submit_bio(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2573)
__submit_bio() (/home/panard/linux-4.19.100/fs/f2fs/data.c:307)
f2fs_mpage_readpages(struct address_space * mapping, struct list_head * pages, struct
f2fs_read_data_pages(struct file * file, struct address_space * mapping, struct list_h
read_pages(struct address_space * mapping, struct file * filp, struct list_head * page
__do_page_cache_readahead(struct address_space * mapping, struct file * filp, unsigned
ra_submit() (/home/panard/linux-4.19.100/mm/internal.h:66)
ondemand_readahead(struct address_space * mapping, struct file_ra_state * ra, struct f
page_cache_async_readahead(struct address_space * mapping, struct file_ra_state * ra,
generic_file_buffered_read(ssize_t written) (/home/panard/linux-4.19.100/mm/filemap.c:
generic_file_read_iter(struct kiocb * iocb, struct iov_iter * iter) (/home/panard/linu
call_read_iter() (/home/panard/linux-4.19.100/include/linux/fs.h:1814)
new_sync_read() (/home/panard/linux-4.19.100/fs/read_write.c:406)
__vfs_read(struct file * file, char * buf, size_t count, loff_t * pos) (/home/panard/l
vfs_read(struct file * file, char * buf, size_t count, loff_t * pos) (/home/panard/lin
ksys_read(unsigned int fd, char * buf, size_t count) (/home/panard/linux-4.19.100/fs/r
__do_sys_read() (/home/panard/linux-4.19.100/fs/read_write.c:589)
__se_sys_read() (/home/panard/linux-4.19.100/fs/read_write.c:587)
```

```
__x64_sys_read(const struct pt_regs * regs) (/home/panard/linux-4.19.100/fs/read_write
do_syscall_64(unsigned long nr, struct pt_regs * regs) (/home/panard/linux-4.19.100/ar
entry_SYSCALL_64() (/home/panard/linux-4.19.100/arch/x86/entry/entry_64.S:238)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

write

writeback 与readahead区别

Write函数通过调用系统调用接口，将数据从应用层copy到内核层，所以write会触发内核态/用户态切换。当数据到达page cache后，内核并不会立即把数据往下传递。而是返回用户空间。数据什么时候写入硬盘，有内核IO调度决定，所以write是一个异步调用。这一点和read不同，read调用是先检查page cache里面是否有数据，如果有，就取出来返回用户，如果没有，就同步传递下去并等待有数据，再返回用户，所以read是一个同步过程。当然你也可以把write的异步过程改成同步过程，就是在open文件的时候带上O_SYNC标记。

O_DIRECT & RAW flag (权重方式的io控制器只能作用于direct io)

O_DIRECT 和 RAW设备最根本的区别是O_DIRECT是基于文件系统的，也就是在应用层来看，其操作对象是文件句柄，内核和文件层来看，其操作是基于inode和数据块，这些概念都是和ext2/3的文件系统相关，写到磁盘上最终是ext3文件。

而RAW设备写是没有文件系统概念，操作的是扇区号，操作对象是扇区，写出来的东西不一定是ext3文件（如果按照ext3规则写就是ext3文件）。

一般基于O_DIRECT来设计优化自己的文件模块，是不满系统的cache和调度策略，自己在应用层实现这些，来制定自己特有的业务特色文件读写。但是写出来的东西是ext3文件，该磁盘卸下来，mount到其他任何linux系统上，都可以查看。

而基于RAW设备的设计系统，一般是不满现有ext3的诸多缺陷，设计自己的文件系统。自己设计文件布局和索引方式。举个极端例子：把整个磁盘做一个文件来写，不要索引。这样没有inode限制，没有文件大小限制，磁盘有多大，文件就能多大。这样的磁盘卸下来，mount到其他linux系统上，是无法识别其数据的。

两者都要通过驱动层读写；在系统引导启动，还处于实模式的时候，可以通过bios接口读写raw设备。

写操作层次关系

具体文件系统层

```
    当具体文件系统层（像ext2/3/4等，我称之为具体文件系统）接到写IO请求时，会判断该IO是否具有DIRECTI
```

static ssize_t ext4_file_write(struct kiocb *iocb,const struct iovec *iov, unsignedlong nr_segs, loff_t pos) { structinode *inode = file_inode(iocb->ki_filp); ssize_tret; … if(unlikely(iocb->ki_filp->f_flags & O_DIRECT)) ret =ext4_file_dio_write(iocb, iov, nr_segs, pos); else ret =generic_file_aio_write(iocb, iov, nr_segs, pos);

returnret; }

直接IO：

```
控制流，若进入直接IO，则调用具体文件系统层的直接IO处理函数，然后调用通过submit_bio函数将IO提交

数据流，数据依旧存放在用户态缓存中，并不需要将数据复制到pagecache中，减少了数据复制次数。
```

缓存IO：

```
控制流，若进入BufferIO，则调用具体文件系统的write_begin进入的准备：比如空间分配，缓存映射(涉及

数据流，数据从用户态复制到内核态page cache中。
```

Page Cache层

```
Page Cache是文件数据在内存中的副本，因此Page Cache管理与内存管理系统和文件系统都相关：一方面Pa
```

通用块层

```
通用块层：由于绝大多数情况的IO操作是跟块设备打交道，所以Linux在此提供了一个类似vfs层的块设备操作

无论是DirectIO还是BufferIO，最后都会通过submit_bio()将IO请求提交到通用块层，通过generic_ma

submit_bio函数通过generic_make_request转发bio，generic_make_request是一个循环，其通过每

Generic_make_request的执行上下文可能有两种，一种是用户上下文，另一种为pdflush所在的内核线程上

在通用块层，提供了一个通用的请求队列压栈方法：blk_queue_bio。在初始化一个有queue块设备驱动的时
```

IO调度层

```
IO调度层：因为绝大多数的块设备都是类似磁盘这样的设备，所以有必要根据这类设备的特点以及应用的不同特

到目前为止，文件系统（pdflush或者address_space_operations）发下来的bio已经merge到request

如果为sync bio，那么直接调用__generic_unplug_device，否则需要在unplug timer的软中断上下文
```

```
    queue是块设备的驱动程序提供的一个请求队列。make_request_fn函数将bio放入请求队列中进行调度处理。

    举例中的sda是一个scsi设备，在scsi middle level将scsi_request_fn函数注册到了queue队列的req
```

设备驱动层

```
    设备驱动程序要做的事情就是从request_queue里面取出请求，然后操作硬件设备，逐个去执行这些请求。除
```

以sisc设备为例：

接下来的过程实际上和具体的scsi总线操作相关了。在scsi_request_fn函数中会扫描request队列，通过elv_next_request函数从队列中获取一个request。在elv_next_request函数中通过scsi总线层注册的q->prep_rq_fn（scsi层注册为scsi_prep_fn）函数将具体的request转换成scsi驱动所能认识的scsi command。获取一个request之后，scsi_request_fn函数直接调用scsi_dispatch_cmd函数将scsi command发送给一个具体的scsi host。到这一步，有一个问题：scsi command具体转发给那个scsi host呢？秘密就在于q->queuedata中，在为sda设备分配queue队列时，已经指定了sda块设备与底层的scsi设备（scsidevice）之间的关系，他们的关系是通过request queue维护的。

```
    在scsi_dispatch_cmd函数中，通过scsi host的接口方法queuecommand将scsi command发送给scsi

     在SCSi中断下半部中，调用scsi command结束的回调函数，这个函数往往为scsi_done，在scsi_done函

    经设备驱动层，将数据复制到disk cache中。
```

f2fs文件系统write操作到generic_make_request的调用栈

```
/*************************************************************************************
generic_make_request(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2
submit_bio(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2573)
__submit_bio() (/home/panard/linux-4.19.100/fs/f2fs/data.c:307)
__submit_merged_bio(struct f2fs_bio_info * io) (/home/panard/linux-4.19.100/fs/f2fs/da
f2fs_submit_page_write(struct f2fs_io_info * fio) (/home/panard/linux-4.19.100/fs/f2fs
do_write_page(struct f2fs_summary * sum, struct f2fs_io_info * fio) (/home/panard/linu
f2fs_outplace_write_data(struct dnode_of_data * dn, struct f2fs_io_info * fio) (/home/
f2fs_do_write_data_page(struct f2fs_io_info * fio) (/home/panard/linux-4.19.100/fs/f2f
__write_data_page(struct page * page, bool * submitted, struct writeback_control * wbc
f2fs_write_cache_pages(struct address_space * mapping, struct writeback_control * wbc,
__f2fs_write_data_pages() (/home/panard/linux-4.19.100/fs/f2fs/data.c:2217)
f2fs_write_data_pages(struct address_space * mapping, struct writeback_control * wbc)
do_writepages(struct address_space * mapping, struct writeback_control * wbc) (/home/p
__writeback_single_inode(struct inode * inode, struct writeback_control * wbc) (/home/
writeback_sb_inodes(struct super_block * sb, struct bdi_writeback * wb, struct wb_writ
__writeback_inodes_wb(struct bdi_writeback * wb, struct wb_writeback_work * work) (/ho
wb_writeback(struct bdi_writeback * wb, struct wb_writeback_work * work) (/home/panard
wb_check_start_all() (/home/panard/linux-4.19.100/fs/fs-writeback.c:1936)
wb_do_writeback() (/home/panard/linux-4.19.100/fs/fs-writeback.c:1962)
wb_workfn(struct work_struct * work) (/home/panard/linux-4.19.100/fs/fs-writeback.c:19
process_one_work(struct worker * worker, struct work_struct * work) (/home/panard/linu
worker_thread(void * __worker) (/home/panard/linux-4.19.100/kernel/workqueue.c:2296)
kthread(void * _create) (/home/panard/linux-4.19.100/kernel/kthread.c:246)
ret_from_fork() (/home/panard/linux-4.19.100/arch/x86/entry/entry_64.S:415)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

f2fs write cgroup weight/f2fs文件系统写操作权重方式io控制器调用栈

```
/*************************************************************************************
cfq_group_served(struct cfq_data * cfqd, struct cfq_group * cfqg, struct cfq_queue * c
__cfq_slice_expired(struct cfq_data * cfqd, struct cfq_queue * cfqq, bool timed_out) (
cfq_slice_expired() (/home/panard/linux-4.19.100/block/cfq-iosched.c:2756)
cfq_select_queue() (/home/panard/linux-4.19.100/block/cfq-iosched.c:3388)
cfq_dispatch_requests(struct request_queue * q, int force) (/home/panard/linux-4.19.10
elv_next_request() (/home/panard/linux-4.19.100/block/blk-core.c:2857)
blk_peek_request(struct request_queue * q) (/home/panard/linux-4.19.100/block/blk-core
scsi_request_fn(struct request_queue * q) (/home/panard/linux-4.19.100/drivers/scsi/sc
__blk_run_queue_uncond() (/home/panard/linux-4.19.100/block/blk-core.c:471)
__blk_run_queue(struct request_queue * q) (/home/panard/linux-4.19.100/block/blk-core.
queue_unplugged(struct request_queue * q, unsigned int depth, bool from_schedule) (/ho
blk_flush_plug_list(struct blk_plug * plug, bool from_schedule) (/home/panard/linux-4.
blk_queue_bio(struct request_queue * q, struct bio * bio) (/home/panard/linux-4.19.100
generic_make_request(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2
submit_bio(struct bio * bio) (/home/panard/linux-4.19.100/block/blk-core.c:2573)
__submit_bio() (/home/panard/linux-4.19.100/fs/f2fs/data.c:307)
__submit_merged_bio(struct f2fs_bio_info * io) (/home/panard/linux-4.19.100/fs/f2fs/da
f2fs_submit_page_write(struct f2fs_io_info * fio) (/home/panard/linux-4.19.100/fs/f2fs
do_write_page(struct f2fs_summary * sum, struct f2fs_io_info * fio) (/home/panard/linu
f2fs_outplace_write_data(struct dnode_of_data * dn, struct f2fs_io_info * fio) (/home/
f2fs_do_write_data_page(struct f2fs_io_info * fio) (/home/panard/linux-4.19.100/fs/f2f
__write_data_page(struct page * page, bool * submitted, struct writeback_control * wbc
f2fs_write_cache_pages(struct address_space * mapping, struct writeback_control * wbc,
__f2fs_write_data_pages() (/home/panard/linux-4.19.100/fs/f2fs/data.c:2217)
f2fs_write_data_pages(struct address_space * mapping, struct writeback_control * wbc)
do_writepages(struct address_space * mapping, struct writeback_control * wbc) (/home/p
__writeback_single_inode(struct inode * inode, struct writeback_control * wbc) (/home/
writeback_sb_inodes(struct super_block * sb, struct bdi_writeback * wb, struct wb_writ
__writeback_inodes_wb(struct bdi_writeback * wb, struct wb_writeback_work * work) (/ho
wb_writeback(struct bdi_writeback * wb, struct wb_writeback_work * work) (/home/panard
wb_check_background_flush() (/home/panard/linux-4.19.100/fs/fs-writeback.c:1880)
wb_do_writeback() (/home/panard/linux-4.19.100/fs/fs-writeback.c:1968)
wb_workfn(struct work_struct * work) (/home/panard/linux-4.19.100/fs/fs-writeback.c:19
process_one_work(struct worker * worker, struct work_struct * work) (/home/panard/linu
worker_thread(void * __worker) (/home/panard/linux-4.19.100/kernel/workqueue.c:2296)
kthread(void * _create) (/home/panard/linux-4.19.100/kernel/kthread.c:246)
```

```
ret_from_fork() (/home/panard/linux-4.19.100/arch/x86/entry/entry_64.S:415)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

```
/*************************************************************************************
    balance_dirty_pages_ratelimited
        f2fs_update_time
        f2fs_put_page
        set_page_dirty
    f2fs_write_end
    flush_dcache_page
    iov_iter_copy_from_user_atomic
        f2fs_wait_on_page_writeback
            __up_read
        __do_map_lock
        if f2fs_has_inline_data  return 0;
        prepare_write_begin //we already allocated all the blocks, so we don't need to
            find_get_entry
        pagecache_get_page(struct address_space * mapping, unsigned long offset, int f
        f2fs_pagecache_get_page() (/home/panard/linux-4.19.100/fs/f2fs/f2fs.h:2044)
    f2fs_write_begin(struct file * file, struct address_space * mapping, loff_t pos, u
generic_perform_write(struct file * file, struct iov_iter * i, loff_t pos) (/home/pana
__generic_file_write_iter(struct kiocb * iocb, struct iov_iter * from) (/home/panard/l
f2fs_file_write_iter(struct kiocb * iocb, struct iov_iter * from) (/home/panard/linux-
call_write_iter() (/home/panard/linux-4.19.100/include/linux/fs.h:1820)
new_sync_write() (/home/panard/linux-4.19.100/fs/read_write.c:474)
__vfs_write(struct file * file, const char * p, size_t count, loff_t * pos) (/home/pan
vfs_write(struct file * file, const char * buf, size_t count, loff_t * pos) (/home/pan
ksys_write(unsigned int fd, const char * buf, size_t count) (/home/panard/linux-4.19.1
__do_sys_write() (/home/panard/linux-4.19.100/fs/read_write.c:611)
__se_sys_write() (/home/panard/linux-4.19.100/fs/read_write.c:608)
__x64_sys_write(const struct pt_regs * regs) (/home/panard/linux-4.19.100/fs/read_writ
do_syscall_64(unsigned long nr, struct pt_regs * regs) (/home/panard/linux-4.19.100/ar
entry_SYSCALL_64() (/home/panard/linux-4.19.100/arch/x86/entry/entry_64.S:238)
```

伪代码, 结构体&函数栈

```
struct address_space {
    struct inode            *host;      /* owner: inode, block_device */若为空，swap ar
    struct radix_tree_root  i_pages;    /* cached pages */
    struct rb_root_cached   i_mmap;     /* tree of private and shared mappings */
    /* Protected by the     i_pages lock */
    unsigned long           nrpages;    /* number of total pages */
    pgoff_t                 writeback_index;/* writeback starts here */
    const struct address_space_operations *a_ops;    /* methods */
    /*指向操作函数表（struct address_space_operations），每个后备存储都要实现这个函数表，比如ex
    unsigned long           flags;      /* error bits */
    spinlock_t              private_lock;    /* for use by the address_space */
    gfp_t                   gfp_mask;   /* implicit gfp mask for allocations */
    struct list_head        private_list;    /* for use by the address_space */
    void                    *private_data;  /* ditto */
    errseq_t                wb_err;
};

const struct address_space_operations f2fs_dblock_aops = {
    .readpage   = f2fs_read_data_page,
    /* readpage()首先会调用find_get_page(mapping, index)在page cache中寻找请求的数据，mappi
    .writepage  = f2fs_write_data_page,
    /* 对于文件映射（host指向一个inode对象），page每次修改后都会调用SetPageDirty（page）将page
    .write_begin    = f2fs_write_begin,
    .write_end  = f2fs_write_end,
    .set_page_dirty = f2fs_set_data_page_dirty,
};

f2fs_read_inline_data : ... ? f2fs_mpage_readpages

f2fs_write_data_page
    --> __write_data_page
        --> f2fs_has_inline_data
            no inline data:
            --> f2fs_do_write_data_page   ???
                --> encrypt_one_page
                --> set_page_writeback
                    -->test_set_page_writeback (page-writeback.c)
                        --> lock_page_memcg(page)
                        --> TestSetPageWriteback
                --> f2fs_inplace_write_data
                    --> f2fs_submit_page_bio
                    --> update_device_state
                    --> f2fs_update_iostat
                --> inode_dec_dirty_pages
                --> f2fs_submit_merged_write_cond
                --> clear_inode_flag
                --> f2fs_remove_dirty_inode
            has inline data:
            --> f2fs_write_inline_data
                --> set_new_dnode
                --> f2fs_wait_on_page_writeback
                --> set_page_dirty


/*当一个block被读入内存或者等待写入块设备时，保存在buffer中，一个buffer对应一个block*/
struct buffer_head {
    unsigned long b_state;      /* buffer state bitmap (see above)  如是否dirty，是否正在
    struct buffer_head *b_this_page;/* circular list of page's buffers */
    struct page *b_page;        /* the page this bh is mapped to 指向buffer所在的page（

    sector_t b_blocknr;     /* start block number */
    size_t b_size;          /* size of mapping  表示block的大小。该block起始于b_data,终止
    char *b_data;           /* pointer to data within the page 直接指向block所在位置（在b_

    struct block_device *b_bdev; /* 指向buffer对应的block所在的块设备对象 */
     void *b_private;       /* reserved for b_end_io */
    struct list_head b_assoc_buffers; /* associated with another mapping */
    struct address_space *b_assoc_map;  /* mapping this buffer is associated with */
    atomic_t b_count;       /* users using this buffer_head buffer的使用计数，调用get_bh(
};

/* 表示一个正在进行的块I/O操作 */
struct bio {
    sector_t bi_sector; /* associated sector on disk */
    struct bio *bi_next; /* list of requests */
    struct block_device *bi_bdev; /* associated block device */
    unsigned long bi_flags; /* status and command flags */
    unsigned long bi_rw; /* read or write? */
    unsigned short bi_vcnt; /* number of bio_vecs off */ /* 是bi_io_vec数组的长度 */
    unsigned short bi_idx; /* current index in bi_io_vec */ /* 当前正在进行I/O操作bio_vec
    unsigned short bi_phys_segments; /* number of segments */
    unsigned int bi_size; /* I/O count */
    unsigned int bi_seg_front_size; /* size of first segment */
    unsigned int bi_seg_back_size; /* size of last segment */
    unsigned int bi_max_vecs; /* maximum bio_vecs possible */
    unsigned int bi_comp_cpu; /* completion CPU */
    atomic_t bi_cnt; /* usage counter */
    struct bio_vec *bi_io_vec; /* bio_vec list */ /* bi_io_vec是一个指针，指向一个bio_vec
    /* bi_io_vec数组使得bio结构体可以支持在一次I/O操作中，使用多个在内存中不连续的segment，这个又
    bio_end_io_t *bi_end_io; /* I/O completion method */
    void *bi_private; /* owner-private method */
    bio_destructor_t *bi_destructor; /* destructor method */
    struct bio_vec bi_inline_vecs[0]; /* inline bio vectors */
};


struct bio_vec {
    struct page *bv_page; //向segment所在的物理page
    unsigned int bv_len; //segment的大小（字节）
    unsigned int bv_offset;//segment起始点在page中的偏移量。
};
```

latency方式io控制器调用栈

author comments

```
Block rq-qos base io controller

This works similar to wbt with a few exceptions

- It's bio based, so the latency covers the whole block layer in addition to the actua
    -它是基于bio的，所以除了实际的io，latency覆盖了整个block层。
- We will throttle all IO that comes in here if we need to.
    -如果需要的话，我们会限制所有进入这里的IO。
- We use the mean latency over the 100ms window.  This is because writes can be partic
    -我们使用100毫秒窗口的平均延迟。这是因为写入速度特别快，这可能会让我们错误地感觉到其他工作负载对受
- By default there's no throttling, we set the queue_depth to INT_MAX so that we can h
    -默认情况下没有限制，我们将队列深度设置为INT_MAX，这样我们就可以拥有尽可能多的未完成的bio。只有
```

```
The hierarchy works like the cpu controller does, we track the latency at every config
层次结构的工作方式与cpu控制器类似，我们跟踪每个配置节点的延迟，每个配置节点都有自己独立的队列深度。这意

Consider the following

                  root blkg
            /                     \
       fast (target=5ms)     slow (target=10ms)
        /     \                  /        \
      a        b          normal(15ms)   unloved

"a" and "b" have no target, but their combined io under "fast" cannot exceed an averag
“a”和“b”没有目标，但它们在“fast”下的组合io不能超过平均5毫秒的延迟，如果超过，我们将限制“slow”组。对

In this example "fast", "slow", and "normal" will be the only groups actually accounti
在本例中，“fast”、“slow”和“normal”将是唯一实际计算io延迟的组。每次提交时，我们都必须从继承人那里走

There are 2 ways we throttle IO.

1) Queue depth throttling.  As we throttle down we will adjust the maximum number of I
1） 队列深度限制。当我们减速时，我们将调整允许在飞行中使用的IO的最大数量。从（u64）-1到1。如果组只为

2) Induced delay throttling.  This is for the case that a group is generating IO that
2） 诱导延迟节流。这是针对一个组正在生成IO的情况，该IO必须由根cg发出以避免优先级反转。所以想想REQ_ME

total_time += min_lat_nsec - actual_io_completion

and then at throttle time will do

throttle_time = min(total_time, NSEC_PER_SEC)

This induced delay will throttle back the activity that is generating the root cg issu
这种诱导的延迟将抑制生成根cg发出的io的活动，不管是一些元数据密集型操作还是组使用了太多内存，以至于将我
```

latency ops:

```
static struct rq_qos_ops blkcg_iolatency_ops = {
    .throttle = blkcg_iolatency_throttle,
    .done_bio = blkcg_iolatency_done_bio,
    .exit = blkcg_iolatency_exit,
};
```

blk_iolatency_init (PS: add loop device)

```
    timer_setup  (blkiolatency_timer_fn)
    blkcg_activate_policy (blkcg_policy_iolatency)
    rq_qos_add (blkcg_iolatency_ops)
blk_iolatency_init(struct request_queue * q) (/root/linux-4.19.100/block/blk-iolatency
blkcg_init_queue(struct request_queue * q) (/root/linux-4.19.100/block/blk-cgroup.c:12
blk_alloc_queue_node(gfp_t gfp_mask, int node_id, spinlock_t * lock) (/root/linux-4.19
blk_mq_init_queue(struct blk_mq_tag_set * set) (/root/linux-4.19.100/block/blk-mq.c:24
loop_add(struct loop_device ** l, int i) (/root/linux-4.19.100/drivers/block/loop.c:19
loop_init() (/root/linux-4.19.100/drivers/block/loop.c:2236)
do_one_initcall(initcall_t fn) (/root/linux-4.19.100/init/main.c:883)
do_initcall_level() (/root/linux-4.19.100/init/main.c:951)
do_initcalls() (/root/linux-4.19.100/init/main.c:959)
do_basic_setup() (/root/linux-4.19.100/init/main.c:977)
kernel_init_freeable() (/root/linux-4.19.100/init/main.c:1144)
kernel_init(void * unused) (/root/linux-4.19.100/init/main.c:1061)
ret_from_fork() (/root/linux-4.19.100/arch/x86/entry/entry_64.S:415)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

blk_iolatency_init (PS: scsi_add_device)

```
    timer_setup  (blkiolatency_timer_fn)
    blkcg_activate_policy (blkcg_policy_iolatency)
    rq_qos_add (blkcg_iolatency_ops)
blk_iolatency_init(struct request_queue * q) (/root/linux-4.19.100/block/blk-iolatency
blkcg_init_queue(struct request_queue * q) (/root/linux-4.19.100/block/blk-cgroup.c:12
blk_alloc_queue_node(gfp_t gfp_mask, int node_id, spinlock_t * lock) (/root/linux-4.19
scsi_old_alloc_queue(struct scsi_device * sdev) (/root/linux-4.19.100/drivers/scsi/scs
scsi_alloc_sdev(struct scsi_target * starget, u64 lun, void * hostdata) (/root/linux-4
scsi_probe_and_add_lun(struct scsi_target * starget, u64 lun, blist_flags_t * bflagsp,
__scsi_add_device(struct Scsi_Host * shost, uint channel, uint id, u64 lun, void * hos
ata_scsi_scan_host(struct ata_port * ap, int sync) (/root/linux-4.19.100/drivers/ata/l
async_port_probe(void * data, async_cookie_t cookie) (/root/linux-4.19.100/drivers/ata
async_run_entry_fn(struct work_struct * work) (/root/linux-4.19.100/kernel/async.c:127
process_one_work(struct worker * worker, struct work_struct * work) (/root/linux-4.19.
worker_thread(void * __worker) (/root/linux-4.19.100/kernel/workqueue.c:2296)
kthread(void * _create) (/root/linux-4.19.100/kernel/kthread.c:246)
ret_from_fork() (/root/linux-4.19.100/arch/x86/entry/entry_64.S:415)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

iolatency_set_min_lat_nsec 设置latency target

```
    iolat->min_lat_nsec = val; //target*1000 (ns)
    iolat->cur_win_nsec = max_t(u64, val << 4, BLKIOLATENCY_MIN_WIN_SIZE) /* 100ms <--
    iolat->cur_win_nsec
iolatency_set_min_lat_nsec(struct blkcg_gq * blkg, u64 val) (/root/linux-4.19.100/bloc
iolatency_set_limit(struct kernfs_open_file * of, char * buf, size_t nbytes, loff_t of
cgroup_file_write(struct kernfs_open_file * of, char * buf, size_t nbytes, loff_t off)
kernfs_fop_write(struct file * file, const char * user_buf, size_t count, loff_t * ppo
__vfs_write(struct file * file, const char * p, size_t count, loff_t * pos) (/root/lin
vfs_write(struct file * file, const char * buf, size_t count, loff_t * pos) (/root/lin
ksys_write(unsigned int fd, const char * buf, size_t count) (/root/linux-4.19.100/fs/r
__do_sys_write() (/root/linux-4.19.100/fs/read_write.c:611)
__se_sys_write() (/root/linux-4.19.100/fs/read_write.c:608)
__x64_sys_write(const struct pt_regs * regs) (/root/linux-4.19.100/fs/read_write.c:608
do_syscall_64(unsigned long nr, struct pt_regs * regs) (/root/linux-4.19.100/arch/x86/
entry_SYSCALL_64() (/root/linux-4.19.100/arch/x86/entry/entry_64.S:238)
```

rq_qos_throttle 调用栈

```
__blkcg_iolatency_throttle
    bio_issue_init
blkcg_iolatency_throttle
rq_qos_throttle(struct request_queue * q, struct bio * bio, spinlock_t * lock) (/root/
blk_queue_bio(struct request_queue * q, struct bio * bio) (/root/linux-4.19.100/block/
generic_make_request(struct bio * bio) (/root/linux-4.19.100/block/blk-core.c:2471)
submit_bio(struct bio * bio) (/root/linux-4.19.100/block/blk-core.c:2580)
__submit_bio() (/root/linux-4.19.100/fs/f2fs/data.c:307)
f2fs_mpage_readpages(struct address_space * mapping, struct list_head * pages, struct
f2fs_read_data_pages(struct file * file, struct address_space * mapping, struct list_h
read_pages(struct address_space * mapping, struct file * filp, struct list_head * page
__do_page_cache_readahead(struct address_space * mapping, struct file * filp, unsigned
ra_submit() (/root/linux-4.19.100/mm/internal.h:66)
do_sync_mmap_readahead() (/root/linux-4.19.100/mm/filemap.c:2467)
filemap_fault(struct vm_fault * vmf) (/root/linux-4.19.100/mm/filemap.c:2543)
f2fs_filemap_fault(struct vm_fault * vmf) (/root/linux-4.19.100/fs/f2fs/file.c:42)
__do_fault(struct vm_fault * vmf) (/root/linux-4.19.100/mm/memory.c:3269)
do_read_fault() (/root/linux-4.19.100/mm/memory.c:3681)
do_fault() (/root/linux-4.19.100/mm/memory.c:3810)
handle_pte_fault() (/root/linux-4.19.100/mm/memory.c:4041)
__handle_mm_fault(struct vm_area_struct * vma, unsigned long address, unsigned int fla
handle_mm_fault(struct vm_area_struct * vma, unsigned long address, unsigned int flags
__do_page_fault(struct pt_regs * regs, unsigned long error_code, unsigned long address
do_page_fault(struct pt_regs * regs, unsigned long error_code) (/root/linux-4.19.100/a
page_fault() (/root/linux-4.19.100/arch/x86/entry/entry_64.S:1204)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
irq_stack_union (Unknown Source:0)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

blkcg_iolatency_done_bio 调用栈

```
    iolatency_record_time
blkcg_iolatency_done_bio(struct rq_qos * rqos, struct bio * bio) (/root/linux-4.19.100
rq_qos_done_bio(struct request_queue * q, struct bio * bio) (/root/linux-4.19.100/bloc
bio_endio(struct bio * bio) (/root/linux-4.19.100/block/bio.c:1755)
req_bio_endio() (/root/linux-4.19.100/block/blk-core.c:285)
blk_update_request(struct request * req, blk_status_t error, unsigned int nr_bytes) (/
scsi_end_request(struct request * req, blk_status_t error, unsigned int bytes, unsigne
scsi_io_completion(struct scsi_cmnd * cmd, unsigned int good_bytes) (/root/linux-4.19.
scsi_finish_command(struct scsi_cmnd * cmd) (/root/linux-4.19.100/drivers/scsi/scsi.c:
scsi_softirq_done(struct request * rq) (/root/linux-4.19.100/drivers/scsi/scsi_lib.c:1
blk_done_softirq(struct softirq_action * h) (/root/linux-4.19.100/block/blk-softirq.c:
__do_softirq() (/root/linux-4.19.100/kernel/softirq.c:292)
```

```
invoke_softirq() (/root/linux-4.19.100/kernel/softirq.c:372)
irq_exit() (/root/linux-4.19.100/kernel/softirq.c:412)
exiting_irq() (/root/linux-4.19.100/arch/x86/include/asm/apic.h:536)
do_IRQ(struct pt_regs * regs) (/root/linux-4.19.100/arch/x86/kernel/irq.c:258)
common_interrupt() (/root/linux-4.19.100/arch/x86/entry/entry_64.S:670)
init_thread_union (Unknown Source:0)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

blk_clear_congested 调用栈

```
blk_clear_congested(struct request_list * rl, int sync) (/root/linux-4.19.100/block/bl
__freed_request(struct request_list * rl, int sync) (/root/linux-4.19.100/block/blk-co
freed_request(struct request_list * rl, bool sync, req_flags_t rq_flags) (/root/linux-
__blk_put_request(struct request_queue * q, struct request * req) (/root/linux-4.19.10
blk_finish_request(struct request * req, blk_status_t error) (/root/linux-4.19.100/blo
scsi_end_request(struct request * req, blk_status_t error, unsigned int bytes, unsigne
scsi_io_completion(struct scsi_cmnd * cmd, unsigned int good_bytes) (/root/linux-4.19.
scsi_finish_command(struct scsi_cmnd * cmd) (/root/linux-4.19.100/drivers/scsi/scsi.c:
scsi_softirq_done(struct request * rq) (/root/linux-4.19.100/drivers/scsi/scsi_lib.c:1
blk_done_softirq(struct softirq_action * h) (/root/linux-4.19.100/block/blk-softirq.c:
__do_softirq() (/root/linux-4.19.100/kernel/softirq.c:292)
invoke_softirq() (/root/linux-4.19.100/kernel/softirq.c:372)
irq_exit() (/root/linux-4.19.100/kernel/softirq.c:412)
exiting_irq() (/root/linux-4.19.100/arch/x86/include/asm/apic.h:536)
do_IRQ(struct pt_regs * regs) (/root/linux-4.19.100/arch/x86/kernel/irq.c:258)
common_interrupt() (/root/linux-4.19.100/arch/x86/entry/entry_64.S:670)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

__mark_inode_dirty 调用栈

```
__mark_inode_dirty(struct inode * inode, int flags) (/root/linux-4.19.100/fs/fs-writeb
mark_inode_dirty_sync() (/root/linux-4.19.100/include/linux/fs.h:2087)
dquot_free_space() (/root/linux-4.19.100/include/linux/quotaops.h:383)
dquot_free_block() (/root/linux-4.19.100/include/linux/quotaops.h:393)
f2fs_i_blocks_write() (/root/linux-4.19.100/fs/f2fs/f2fs.h:2390)
dec_valid_block_count() (/root/linux-4.19.100/fs/f2fs/f2fs.h:1773)
f2fs_truncate_data_blocks_range(struct dnode_of_data * dn, int count) (/root/linux-4.1
f2fs_truncate_data_blocks(struct dnode_of_data * dn) (/root/linux-4.19.100/fs/f2fs/fil
truncate_dnode(struct dnode_of_data * dn) (/root/linux-4.19.100/fs/f2fs/node.c:880)
truncate_nodes(struct dnode_of_data * dn, unsigned int nofs, int ofs, int depth) (/roo
f2fs_truncate_inode_blocks(struct inode * inode, unsigned long from) (/root/linux-4.19
f2fs_truncate_blocks(struct inode * inode, u64 from, bool lock) (/root/linux-4.19.100/
f2fs_truncate(struct inode * inode) (/root/linux-4.19.100/fs/f2fs/file.c:685)
f2fs_setattr(struct dentry * dentry, struct iattr * attr) (/root/linux-4.19.100/fs/f2f
notify_change(struct dentry * dentry, struct iattr * attr, struct inode ** delegated_i
do_truncate(struct dentry * dentry, loff_t length, unsigned int time_attrs, struct fil
handle_truncate() (/root/linux-4.19.100/fs/namei.c:3009)
do_last() (/root/linux-4.19.100/fs/namei.c:3427)
```

```
path_openat(struct nameidata * nd, const struct open_flags * op, unsigned int flags) (
do_filp_open(int dfd, struct filename * pathname, const struct open_flags * op) (/root
do_sys_open(int dfd, const char * filename, int flags, umode_t mode) (/root/linux-4.19
__do_sys_openat(int flags) (/root/linux-4.19.100/fs/open.c:1115)
__se_sys_openat() (/root/linux-4.19.100/fs/open.c:1109)
__x64_sys_openat(const struct pt_regs * regs) (/root/linux-4.19.100/fs/open.c:1109)
do_syscall_64(unsigned long nr, struct pt_regs * regs) (/root/linux-4.19.100/arch/x86/
entry_SYSCALL_64() (/root/linux-4.19.100/arch/x86/entry/entry_64.S:238)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

blkcg_iolatency_throttle 调用栈

```
blkcg_iolatency_throttle(struct rq_qos * rqos, struct bio * bio, spinlock_t * lock) (/
rq_qos_throttle(struct request_queue * q, struct bio * bio, spinlock_t * lock) (/root/
blk_queue_bio(struct request_queue * q, struct bio * bio) (/root/linux-4.19.100/block/
generic_make_request(struct bio * bio) (/root/linux-4.19.100/block/blk-core.c:2471)
submit_bio(struct bio * bio) (/root/linux-4.19.100/block/blk-core.c:2580)
__submit_bio() (/root/linux-4.19.100/fs/f2fs/data.c:307)
__submit_merged_bio(struct f2fs_bio_info * io) (/root/linux-4.19.100/fs/f2fs/data.c:32
f2fs_submit_page_write(struct f2fs_io_info * fio) (/root/linux-4.19.100/fs/f2fs/data.c
do_write_page(struct f2fs_summary * sum, struct f2fs_io_info * fio) (/root/linux-4.19.
f2fs_outplace_write_data(struct dnode_of_data * dn, struct f2fs_io_info * fio) (/root/
f2fs_do_write_data_page(struct f2fs_io_info * fio) (/root/linux-4.19.100/fs/f2fs/data.
__write_data_page(struct page * page, bool * submitted, struct writeback_control * wbc
f2fs_write_cache_pages(struct address_space * mapping, struct writeback_control * wbc,
__f2fs_write_data_pages() (/root/linux-4.19.100/fs/f2fs/data.c:2217)
f2fs_write_data_pages(struct address_space * mapping, struct writeback_control * wbc)
do_writepages(struct address_space * mapping, struct writeback_control * wbc) (/root/l
__writeback_single_inode(struct inode * inode, struct writeback_control * wbc) (/root/
writeback_sb_inodes(struct super_block * sb, struct bdi_writeback * wb, struct wb_writ
__writeback_inodes_wb(struct bdi_writeback * wb, struct wb_writeback_work * work) (/ro
wb_writeback(struct bdi_writeback * wb, struct wb_writeback_work * work) (/root/linux-
wb_check_start_all() (/root/linux-4.19.100/fs/fs-writeback.c:1954)
wb_do_writeback() (/root/linux-4.19.100/fs/fs-writeback.c:1981)
wb_workfn(struct work_struct * work) (/root/linux-4.19.100/fs/fs-writeback.c:2015)
process_one_work(struct worker * worker, struct work_struct * work) (/root/linux-4.19.
worker_thread(void * __worker) (/root/linux-4.19.100/kernel/workqueue.c:2296)
kthread(void * _create) (/root/linux-4.19.100/kernel/kthread.c:246)
ret_from_fork() (/root/linux-4.19.100/arch/x86/entry/entry_64.S:415)
[Unknown/Just-In-Time compiled code] (Unknown Source:0)
```

blkcg_maybe_throttle_blkg 调用栈

```
exit_to_usermode_loop() {
    mem_cgroup_handle_over_high();
    blkcg_maybe_throttle_current() {
        /* blkcg_maybe_throttle_current[1750] ---  */
        kthread_blkcg();
        blkg_lookup_slowpath();
        ktime_get();
        /* blkcg_maybe_throttle_blkg[1689] ---  */
        blkcg_scale_delay();
        blk_put_queue();
    }
}
// TODO: 测试是否会调用到blkcg_maybe_throttle_blkg, 已经添加printk, 检查是否有delay_nsec输出
```

qos 调用栈 (多核cpu虚拟机trace文件截取)

PS: 多核cpu与单核cpu在执行同样的cgroup io动作时，流程似有差异，低优先级研究

```
__brelse();
__getblk_gfp() {
  __find_get_block() {
    pagecache_get_page() {
      find_get_entry();
    }
  }
  _cond_resched() {
    rcu_all_qs();
  }
  __find_get_block() {
    pagecache_get_page() {
      find_get_entry();
    }
  }
  pagecache_get_page() {
    find_get_entry();
    __page_cache_alloc() {
      alloc_pages_current() {
        get_task_policy();
        policy_nodemask();
        policy_node();
        __alloc_pages_nodemask() {
          _cond_resched() {
            rcu_all_qs();
          }
          get_page_from_freelist() {
            __inc_numa_state();
            __inc_numa_state();
          }
        }
      }
    }
    add_to_page_cache_lru() {
      __add_to_page_cache_locked() {
        PageHuge();
        mem_cgroup_try_charge() {
          get_mem_cgroup_from_mm() {
            mem_cgroup_from_task();
          }
          try_charge() {
            do_memsw_account();
            page_counter_try_charge() {
              propagate_protected_usage();
              propagate_protected_usage();
            }
            refill_stock() {
              drain_stock() {
                page_counter_uncharge() {
                  page_counter_cancel() {
                    propagate_protected_usage();
                  }
                  page_counter_cancel() {
                    propagate_protected_usage();
                  }
                }
                do_memsw_account();
              }
            }
          }
        }
        _raw_spin_lock_irq();
        page_cache_tree_insert() {
          shmem_mapping();
          workingset_update_node();
        }
        __inc_node_page_state() {
          __inc_node_state();
        }
        mem_cgroup_commit_charge() {
          mem_cgroup_charge_statistics();
          memcg_check_events();
          do_memsw_account();
        }
      }
      lru_cache_add() {
        __lru_cache_add();
      }
    }
  }
  alloc_page_buffers() {
    get_mem_cgroup_from_page();
    alloc_buffer_head() {
      kmem_cache_alloc() {
        _cond_resched() {
          rcu_all_qs();
        }
        should_failslab();
        memcg_kmem_get_cache();
        memcg_kmem_put_cache();
      }
      recalc_bh_state();
    }
    set_bh_page();
```

```
    }
    _raw_spin_lock();
    init_page_buffers() {
      I_BDEV();
    }
    unlock_page();
    __find_get_block() {
      pagecache_get_page() {
        find_get_entry();
        mark_page_accessed() {
          workingset_activation();
        }
      }
      _raw_spin_lock();
      __brelse();
    }
  }
  ll_rw_block() { // low level access to block devices
    submit_bh() {
      submit_bh_wbc(struct buffer_head *bh, struct writeback_control *wbc) {
        bio_alloc_bioset() {
          mempool_alloc() {
            _cond_resched() {
              rcu_all_qs();
            }
            mempool_alloc_slab() {
              kmem_cache_alloc() {
                should_failslab();
                memcg_kmem_put_cache();
              }
            }
          }
          bio_init();
        }
        bio_add_page() {
          __bio_try_merge_page();
          __bio_add_page();
        }
        guard_bio_eod() { // 这允许我们在设备的奇数个最后扇区上执行IO偶数，即使块大小是物理扇区大小
          __disk_get_part(bio->bi_disk, bio->bi_partno);
        }
        submit_bio(struct bio *bio) { // fs/buffer.c:3094 //submit a bio to the block de
          generic_make_request() { //block/block-core.c:2415
            blk_queue_enter(); // try to increase q->q_usage_counter
            generic_make_request_checks() { //check device exist, sector num, read only.
              _cond_resched() {
                rcu_all_qs();
              }
              should_fail_bio();
              __disk_get_part();
              blk_partition_remap(); // CX____ add
              /* block_bio_remap: 8,0 RA 58724688 + 8 <- (8,2) 58720592 */
              create_io_context(GFP_ATOMIC, q->node); // CX____ add
              blkcg_bio_issue_check () {
                  blkcg = bio_blkcg(bio); {
```

```
          kthread_blkcg();
      }
      bio_associate_blkcg();
      blkg_lookup_slowpath();
      blk_throtl_bio() {
          /* blk_throtl_bio[2148] ---  */
          throtl_update_latency_buckets(td);
          blk_throtl_assoc_bio(tg, bio);
          blk_throtl_update_idletime(tg);
          throtl_downgrade_check(tg);
          throtl_upgrade_check(tg);
          if (throtl_can_upgrade(td, tg)) {
              throtl_upgrade_state(td);
              goto again;
          }
          throtl_charge_bio(tg, bio) {
              bio_set_flag(bio, BIO_THROTTLED);
          }
          throtl_trim_slice(tg, rw);
          throtl_add_bio_tg(bio, qn, tg);
          if (tg->flags & THROTL_TG_WAS_EMPTY) {
              tg_update_disptime(tg);
              throtl_schedule_next_dispatch(tg->service_queue.parent_sq, tru
          }
      }
      blkg_rwstat_add();
  }
  trace_block_bio_queue(); /* block_bio_queue: 8,0 RA 58724688 + 8 [bash] */
}

/*int blk_init_allocated_queue(struct request_queue *q)
      {
          blk_queue_make_request(q, blk_queue_bio);
      }
*/
q->make_request_fn /* aka */ blk_queue_bio() {
  blk_queue_bounce();
  blk_queue_split();
  bio_integrity_prep();
  blk_attempt_plug_merge();
  _raw_spin_lock_irq();
  elv_merge() {
    elv_rqhash_find();
  }
  rq_qos_throttle() {
    blkcg_iolatency_throttle() {
      /* blkcg_iolatency_throttle[439] --- */
      bio_associate_blkcg();
      blkg_lookup_slowpath();
      ktime_get();
      /* CX____ bio_issue_init[139]: issue->value=18014516950275448 */
      bio_associate_blkg();
      /* blkcg_iolatency_throttle[458] --- bio->bi_issue=137408 */
      // TODO: here change the scale, max_depth
      check_scale_change();
```

```
//__blkcg_iolatency_throttle();
/* __blkcg_iolatency_throttle[207] --- use_delay ^^^ use=0 */
// TODO: This set's the notify_resume for the task to check and see if
blkcg_schedule_throttle() {
  blk_get_queue();
  /* blkcg_schedule_throttle[1807] --- current->throttle_queue.id=8 */
  blk_put_queue(); //减小引用?
  current->throttle_queue = q;
    if (use_memdelay)
        current->use_memdelay = use_memdelay;
    set_notify_resume(current); // 向指定的进程设置了一个TIF_NOTIFY_RESUME
}
// TODO: 重要，增加了inflight
iolatency_may_queue() {
    rq_wait_inc_below() {
        /* rq_wait_inc_below[30] --- set rq_weit->inflight to 1 */
        /* atomic_inc_below[13] --- */
    }
}
/* 将iolat->rq_wait->wait添加到等待队列 */
prepare_to_wait_exclusive(TASK_UNINTERRUPTIBLE) {
  _raw_spin_lock_irqsave();
  set_current_state(WQ_FLAG_EXCLUSIVE);
  _raw_spin_unlock_irqrestore();
}
iolatency_may_queue () {
    rq_wait_inc_below() {
        /* rq_wait_inc_below[30] --- set rq_weit->inflight to 1 */
        /* atomic_inc_below[13] --- */
    }
}
/* __blkcg_iolatency_throttle[237] --- io_schedule() */
// TODO: 打印io_schedule()耗时，看当前进程被wait的时间
io_schedule() {
    current->in_iowait = 1;
    /* 当task发生iowait的时候，内核对他们的处理方法是将task切换出去，让可运行的t
    /* 内核在调用io_schedule，io_schedule_timeout前都会设置task运行状态TASK
    /* http://m.elecfans.com/article/611049.html */
    io_schedule_prepare();
      blk_schedule_flush_plug(current);
  schedule() {
    rcu_note_context_switch() {
      rcu_sched_qs();
    }
    _raw_spin_lock();
    update_rq_clock();
    deactivate_task() {
      dequeue_task_fair() {
        dequeue_entity() {
          update_curr() {
            update_min_vruntime();
            cpuacct_charge();
            __cgroup_account_cputime() {
              cgroup_rstat_updated();
            }
```

```
                            }
                            __update_load_avg_se();
                            __update_load_avg_cfs_rq();
                            clear_buddies();
                            account_entity_dequeue();
                            update_cfs_group();
                            update_min_vruntime();
                          }
                          dequeue_entity() {
                            update_curr() {
                              update_min_vruntime();
                            }
                            __update_load_avg_se();
                            __update_load_avg_cfs_rq();
                            clear_buddies();
                            account_entity_dequeue();
                            update_cfs_group() {
                              reweight_entity();
                            }
                            update_min_vruntime();
                          }
                          hrtick_update();
                        }
                      }
                      __delayacct_blkio_start() {
                        ktime_get();
                      }
                      pick_next_task_fair() {
                        check_cfs_rq_runtime();
                        pick_next_entity() {
                          __pick_first_entity();
                          clear_buddies();
                        }
                        pick_next_entity() {
                          __pick_first_entity();
                          clear_buddies();
                        }
                        put_prev_entity() {
                          check_cfs_rq_runtime();
                          check_spread();
                        }
                        set_next_entity() {
                          __update_load_avg_se() {
                            decay_load();
                            decay_load();
                            decay_load();
                            __accumulate_pelt_segments() {
                              decay_load();
                              decay_load();
                            }
                          }
                          __update_load_avg_cfs_rq() {
                            decay_load();
                            decay_load();
                            decay_load();
```

```
                         __accumulate_pelt_segments() {
                           decay_load();
                           decay_load();
                         }
                       }
                     }
                     put_prev_entity() {
                       check_cfs_rq_runtime();
                       check_spread();
                     }
                     set_next_entity() {
                       __update_load_avg_se() {
                         decay_load();
                         decay_load();
                         decay_load();
                         __accumulate_pelt_segments() {
                           decay_load();
                           decay_load();
                         }
                       }
                       __update_load_avg_cfs_rq();
                     }
                   }
                   switch_mm_irqs_off() {
                     load_new_mm_cr3();
                   }
------------------------------------------
0)   sleep-2186   =>  systemd-368
------------------------------------------
------------------------------------------
0)  systemd-387   =>   sleep-2186
------------------------------------------

                   finish_task_switch();
                 } /* schedule */
               } /* io_schedule */
               _raw_spin_lock_irq();
               prepare_to_wait_exclusive() {
                 _raw_spin_lock_irqsave();
                 _raw_spin_unlock_irqrestore();
               }
               // iolatency_may_queue
               rq_wait_inc_below() {
                 /* rq_wait_inc_below[30] --- set rq_weit->inflight to 1 */
                 /* atomic_inc_below[13] --- */
               }
               //break;
               finish_wait() {
                 _raw_spin_lock_irqsave();
                 _raw_spin_unlock_irqrestore();
               }
             } /* blkcg_iolatency_throttle */
           } /* rq_qos_throttle */
           get_request() {
             blkg_lookup_slowpath();
```

```
                elv_may_queue();
                mempool_alloc() {
                  _cond_resched() {
                    rcu_all_qs();
                  }
                  alloc_request_size() {
                    __kmalloc_node() {
                      kmalloc_slab();
                      should_failslab();
                      memcg_kmem_put_cache();
                    }
                    scsi_old_init_rq() {
                      kmem_cache_alloc_node() {
                        should_failslab();
                        memcg_kmem_put_cache();
                      }
                    }
                  }
                }
                blk_rq_init() {
                  ktime_get();
                }
                elv_set_request();
                /* block_getrq: 8,0 RA 58724688 + 8 [bash] */
              }
              rq_qos_track();
              blk_init_request_from_bio() {
                blk_rq_bio_prep();
              }
              _raw_spin_lock_irq();
              add_acct_request() {
                  blk_account_io_start() {
                  disk_map_sector_rcu();
                  part_round_stats();
                  part_inc_in_flight();
                  }
                  __elv_add_request() {
                  /* block_rq_insert: 8,0 RA 4096 () 58724688 + 8 [bash] */
                  elv_rqhash_add();
                  noop_add_request();
                  }
              }
              __blk_run_queue() {
                scsi_request_fn() {
                  blk_peek_request() {
                    noop_dispatch() {
                      elv_dispatch_sort() {
                        elv_rqhash_del();
                      }
                    }
                    /* block_rq_issue: 8,0 RA 4096 () 58724688 + 8 [bash] */
                    scsi_prep_fn() {
                      scsi_prep_state_check();
                      get_device();
                      scsi_init_command() {
                        scsi_initialize_rq() {
                          scsi_req_init();
                        }
                        init_timer_key();
                        scsi_add_cmd_to_list();
                      }
                      scsi_setup_cmnd() {
                        sd_init_command() {
                          scsi_init_io() {
                            scsi_init_sgtable() {
                              mempool_alloc() {
                                mempool_alloc_slab() {
                                  kmem_cache_alloc() {
                                    should_failslab();
                                    memcg_kmem_put_cache();
                                  }
                                }
                              }
                              blk_rq_map_sg();
                            }
                          }
                        }
                      }
                    }
                  }
                  blk_queue_start_tag() {
                    blk_start_request() {
                      ktime_get();
                      rq_qos_issue();
                      blk_add_timer() {
                        round_jiffies_up() {
                          round_jiffies_common();
                        }
                        blk_rq_timeout() {
                          round_jiffies_up() {
                            round_jiffies_common();
                          }
                        }
                      }
                    }
                  }
                  scsi_init_cmd_errh();
                  scsi_dispatch_cmd() {
                    scsi_log_send();
                    ata_scsi_queuecmd() {
                      _raw_spin_lock_irqsave();
                      ata_scsi_find_dev() {
                        __ata_scsi_find_dev() {
                          ata_find_dev();
                        }
                      }
                      ata_scsi_translate() {
                        ata_qc_new_init();
                        ata_sg_init();
                        ata_scsi_rw_xlat() {
```

```
                        ata_check_nblocks();
                        ata_build_rw_tf();
                      }
                      ahci_pmp_qc_defer() {
                        ata_std_qc_defer();
                      }
                      ata_qc_issue() {
                        dma_direct_map_sg() {
                          check_addr();
                        }
                        ahci_qc_prep() {
                          ata_tf_to_fis();
                          ahci_fill_cmd_slot();
                        }
                        ahci_qc_issue();
                      }
                    }
                    _raw_spin_unlock_irqrestore();
                  }
                }
                _raw_spin_lock_irq();
                blk_peek_request() {
                  noop_dispatch();
                }
              }
            }
          } /* blk_queue_bio */
          blk_queue_exit();
        } /* generic_make_request */
      } /* submit_bio */
    } /* submit_bh_wbc */
  } /* submit_bh */
} /* ll_rw_block */
```

structs 相关结构体

```
struct blkcg {
    struct cgroup_subsys_state  css;
    spinlock_t          lock;

    struct radix_tree_root      blkg_tree;
    struct blkcg_gq __rcu       *blkg_hint;
    struct hlist_head       blkg_list;

    struct blkcg_policy_data    *cpd[BLKCG_MAX_POLS];

    struct list_head        all_blkcgs_node;
#ifdef CONFIG_CGROUP_WRITEBACK
    struct list_head        cgwb_list;
    refcount_t          cgwb_refcnt;
#endif
};
```

```
struct blkcg_gq {
    /* Pointer to the associated request_queue */
    struct request_queue        *q;
    struct list_head        q_node;
    struct hlist_node       blkcg_node;
    struct blkcg            *blkcg;

    /*
     * Each blkg gets congested separately and the congestion state is
     * propagated to the matching bdi_writeback_congested.
     */
    struct bdi_writeback_congested  *wb_congested;

    /* all non-root blkcg_gq's are guaranteed to have access to parent */
    struct blkcg_gq         *parent;

    /* request allocation list for this blkcg-q pair */
    struct request_list     rl;

    /* reference count */
    atomic_t            refcnt;

    /* is this blkg online? protected by both blkcg and q locks */
    bool                online;

    struct blkg_rwstat      stat_bytes;
    struct blkg_rwstat      stat_ios;

    struct blkg_policy_data     *pd[BLKCG_MAX_POLS];

    struct rcu_head         rcu_head;

    atomic_t            use_delay;
    atomic64_t          delay_nsec;
    atomic64_t          delay_start;
    u64             last_delay;
    int             last_use;
};

struct blk_iolatency {
    struct rq_qos rqos;
    struct timer_list timer;
    atomic_t enabled;
};

struct bio {
    struct bio      *bi_next;   /* request queue link */
    struct gendisk      *bi_disk;
    unsigned int        bi_opf;     /* bottom bits req flags,
                         * top bits REQ_OP. Use
                         * accessors.
                         */
    unsigned short      bi_flags;   /* status, etc and bvec pool number */
    unsigned short      bi_ioprio;
    unsigned short      bi_write_hint;
    blk_status_t        bi_status;
    u8          bi_partno;

    /* Number of segments in this BIO after
     * physical address coalescing is performed.
     */
    unsigned int        bi_phys_segments;

    /*
     * To keep track of the max segment size, we account for the
     * sizes of the first and last mergeable segments in this bio.
     */
    unsigned int        bi_seg_front_size;
    unsigned int        bi_seg_back_size;

    struct bvec_iter    bi_iter;

    atomic_t        __bi_remaining;
    bio_end_io_t        *bi_end_io;

    void            *bi_private;
#ifdef CONFIG_BLK_CGROUP
    /*
     * Optional ioc and css associated with this bio.  Put on bio
     * release.  Read comment on top of bio_associate_current().
     */
    struct io_context   *bi_ioc;
    struct cgroup_subsys_state *bi_css;
    struct blkcg_gq     *bi_blkg;
    struct bio_issue    bi_issue;
#endif
};

struct iolatency_grp {
    struct blkg_policy_data pd;
    struct blk_rq_stat __percpu *stats;
    struct blk_iolatency *blkiolat;
    struct rq_depth rq_depth;
    struct rq_wait rq_wait;
    atomic64_t window_start;
    atomic_t scale_cookie;
    u64 min_lat_nsec;
    u64 cur_win_nsec;

    /* total running average of our io latency. */
    u64 lat_avg;

    /* Our current number of IO's for the last summation. */
    u64 nr_samples;

    struct child_latency_info child_lat;
};
```

actions

rq to blkg path

blk_qc_t generic_make_request(struct bio *bio) struct request_queue *q = bio->bi_disk->queue; blk_queue_bio(struct request_queue *q, struct bio *bio) rq_qos_throttle(struct request_queue *q, struct bio *bio, spinlock_t *lock) blkcg_iolatency_throttle(struct rq_qos *rqos, struct bio *bio, spinlock_t *lock)

latency qos与wbt的差异

latency qos与wbt实现很相似，但属于两种不同的东西， 启用latency时wbt并未启用，后续列出详细差异

TODO

- [x] 尝试是否能找到适合google pixel 2xl的4.15以上的内核，若能找到，编译一个可开启cgroup v2的版本并测试． //不可行，仅找到4.4内核
- [ ] 继续研究组迁移时的逻辑，若与推测相符，提出修改思路． //重点放在快速改变队列深度，以及队列深度对读写速度的影响
- [ ] 根据对iolatency的已有研究，尝试移植到4.9版本内核上去 // 优先级低, 目前看来可行性不高
- [x] 补充generic_make_request与设备驱动之间的联系 //已经理清，需要总结
- [x] 什么情况下走了wbt， 什么情况下走了delay & io_schedule // wbt 与iolatency为不同的rq_qos, 各自通过rq_qos_add来生效, wbt->BLK_WBT; 二者实现原理相似，互不依赖．
- [x] 研究一下latency_unknown由来， 优先级低 //属于wbt, skip
- [ ] 研究队列深度对组写入速度的影响 //在emmc, ssd, hdd和不周文件系统均有差异，需要分开研究
- [x] 研究一种在一些位置打印出组信息，队列信息，队列深度的方法 //队列深度未完成，rq和rqos队列深度
- [ ] 研究组切换时，能否实时调整组深度，和实时的组深度变化
- [ ] 研究generic_perform_request与pagecache的路径关系
- [x] 研究wb_timer_fn与时间窗口是否对应 //wbt, skip
- [x] emmc流程需要抓log整理，找出一些方法指针的对应 //暂时放弃，seattletmo为ufs， 与x86一样走scsi. 需注意可能存在mq

- [x] 找出所有的request_queue //确定一个磁盘设备一个rq
- [x] request_queue 与cgroup的对应关系，是否一对一 //no, gendisk<->request_queue
- [x] 在某一时刻打出rq里有多少的request
- [x] 在throttle方法中打出当前的rq所属组 //不可行，rq为已经没有组归属，仅有一个root_blkg
- [x] 参考cfq在权重改变时的重排方法 //latency 并不需要，并未在电梯层使用树

- [ ] 在一个组上，depth, inflight代表什么意义
- [x] 跟踪一个bio，弄明白bio的完成机制，是在放入rq时完结，还是被下发到驱动才完结 //done mostly
- [ ] single core不走io_scheduler()路径，why？与smp区别在哪里？ //低优先级研究
- [ ] 走到io_scheduler的具体条件是什么？ // on going
- [ ] nr_samples, samples_thresh, deepth, scale_cookie
- [ ] plug unplug
- [ ] depth & use_delay //全部为随机4k时，应用depth, 没有限流效果 //use_delay在哪里执行的delay? //if (iolat->rq_depth.max_depth == 1 && direction < 0) , use_delay //blkcg_use_delay --> atomic_inc(&blkg->blkcg->css.cgroup->congestion_count); //depth的调整速度是关键，目前看来最需要修改的是这一块。
- [ ] block io 在memory部分的逻辑，cgroup v2相比v1最重要的改变之一就是关联了cache