disk power management

Felsing, Waidler, Ziegler

2012-02-06

overview

1. wfat
    - basic outline
    - writing
        - 1st attempt
        - 2nd attempt
        - 3rd attempt
        - the metadata problem
    - reading
    - lessons learned
2. scsi_power
    - implementation
    - measurements

a hybrid file system

generic idea

- large HDD with high power consumption
- small SSD with low power consumption

$\rightarrow$ large file system with low power consumption

our approach

- SSD and HDD contain the same data
- metadata is only contained on the SSD
- serve read access from HDD, if
  - HDD running
  - accessed data beyond end of SSD

$\Rightarrow$ SSD is "direct mapped cache" of the first 32 GB

basic outline

VFS

- provides a generic API
- dispatches calls to functions of specific file systems
- performs a lot of work for us (like the rest of the kernel)

approach

1. copy VFAT to WFAT
2. ?
3. profit!

1st attempt: duplicating data structures

basic idea

- create a superblock for the second device
- allocate a page
- clone necessary data

the problem

- "too much" bookkeeping on side of the kernel
- crashes when calling `sync`

"how does this buffer cache even work?"

- a buffer represents a block of data on the disk
- a block consists of at least one sector
- blocks are not larger than a page
  $\Rightarrow$ a page holds one to eight buffers
- a `buffer_head` is used to manage a buffer

2nd attempt: can we just set the device?

- `fat_get_block` is called when a buffer is to be allocated
- resulting `buffer_head` has attributes `b_blocknr` and `b_bdev`
- print `b_blocknr` for debugging purposes
- change `b_bdev` to point to the other device
- data seems to have been written to the original device, though
- writes wouldn't have been mirrored anyways

basic idea of mirrored writing

- duplicating data structures wastes memory
- contents of both devices are practically the same
- dirty pages of one device should be written to the other one as well

3rd attempt: modifying `buffer_heads` in `fat_writepage`

- cycle through the `buffer_heads` of the page
- replace the `b_bdev` by either SDD or HDD device
- state of the `buffer_heads` has to be set to dirty again
- call generic `block_write_full_page`
- `dd if=/dev/sdX1 of=sdX1:SECTOR.bin \`
  `skip=SECTOR count=1`
  tells us that we were successful

FAT metadata

- FAT is created at format time
- FAT size is fixed
- $\Rightarrow$ metadata describes size of the device

the metadata problem

- the hybrid fs should have the size of the large device
- metadata reads and writes bypass `fat_writepage`
- if we format and mount the HDD, metadata ends up on the wrong device
- if we format and mount the SSD, the fs is limited in size

solution

- format HDD (sda)
- `dd if=/dev/sda of=/dev/sdb count=2M`
- now we can simply mount the SSD

reading

approach

- reading is done from page cache
- page has to be filled with buffers beforehand
- recall `fat_get_block` is called on every allocation of a buffer
- resulting `buffer_head` can be modified
- adjust `b_bdev` if necessary conditions hold

checking the results

- again, dd is our friend
- overwrite the block of a file on one device
- `cat` tells us that, depending on HDD state, read access is served from either HDD or SDD

lessons learned

- wait for async operations!
- locking and unlocking is done in different layers (interwoven)
- don't trust the names
- getting comfortable with the code takes its time
- documentation is scattered across the source code comments

scsi_power

basic idea

- write kernel module to put HDD to standby
- works together with wfat, read from SSD when HDD in standby

DDT/ES-Algorithmus

Festplatte in den Standby-Modus setzen, falls

$$t_{la} + t_{be} \leq t$$

oder

$$t_{fa} + t_{lb} \leq t \leq t_{fa} + t_{lb} + t_1 \quad \text{und} \quad t_{la} + t_2 \leq t$$

Die Variablen bedeuten:

t : die aktuelle Zeit
t_{lb} : die Länge der letzten aktiven Phase (Dauer eines I/O-Transfers, `busy_period`)
t_{fa} : der Zeitpunkt des ersten Zugriffs in der gerade aktiven Phase (`first_access`)
t_{la} : wann wurde zuletzt auf die Festplatte zugegriffen
t_{be} : die break-even-Zeit
t_1 : Toleranzwert beim Vergleich des letzten mit dem aktuellen Interval (2 Sekunden)
t_2 : kleiner Timeout, um das Ende der aktiven Phase zu erkennen (1 Sekunde)

scsi_power

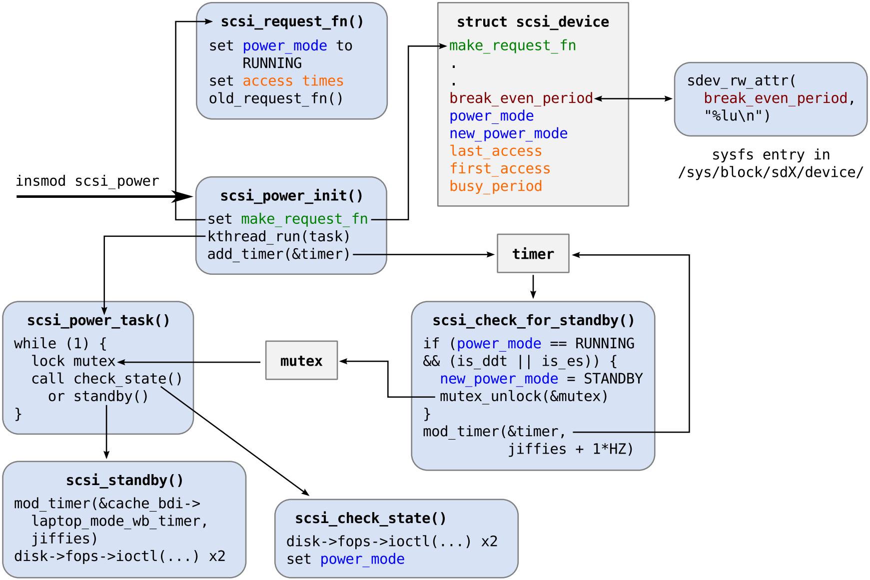
initial stumbling blocks

- IDE kernel module deprecated ⇒ scsi_device
- procfs mainly for process information ⇒ sysfs
- `/sys/block/sda/device/break_even_period`

design decisions

- only changes to kernel code: new fields in scsi_device and added sysfs entry
- all actual code in module

scsi_power


scsi_request_fn()
set power_mode to RUNNING
set access times
old_request_fn()
struct scsi_device
make_request_fn
.
.
break_even_period
power_mode
new_power_mode
last_access
first_access
busy_period
sdev_rw_attr(
break_even_period,
"%lu\n")
sysfs entry in
/sys/block/sdX/device/
insmod scsi_power
scsi_power_init()
set make_request_fn
kthread_run(task)
add_timer(&timer)
timer
scsi_power_task()
while (1) {
lock mutex
call check_state()
or standby()
}
mutex
scsi_check_for_standby()
if (power_mode == RUNNING
&& (is_ddt || is_es)) {
new_power_mode = STANDBY
mutex_unlock(&mutex)
}
mod_timer(&timer,
jiffies + 1*HZ)
scsi_standby()
mod_timer(&cache_bdi->
laptop_mode_wb_timer,
jiffies)
disk->fops->ioctl(...) x2
scsi_check_state()
disk->fops->ioctl(...) x2
set power_mode

scsi_power

more stumbling blocks

- how to recognize disk access?
  in fs/buffer.c:bh_submit() or in actual SCSI driver?
  ⇒ Register a new SCSI device request function
- laptop_mode's writeback didn't work, but why?
  wfat only registers SSD as mounted
  ⇒ SSD has to be passed as device for writeback
- writeback is asynchronous, no synchronization mechanism ⇒

```
set_current_state(TASK_INTERRUPTIBLE);
do {
    msleep(100);
} while(test_bit(BDI_writeback_running,
                 &cache_bdi->state));
set_current_state(TASK_RUNNING);
```

measurement environment

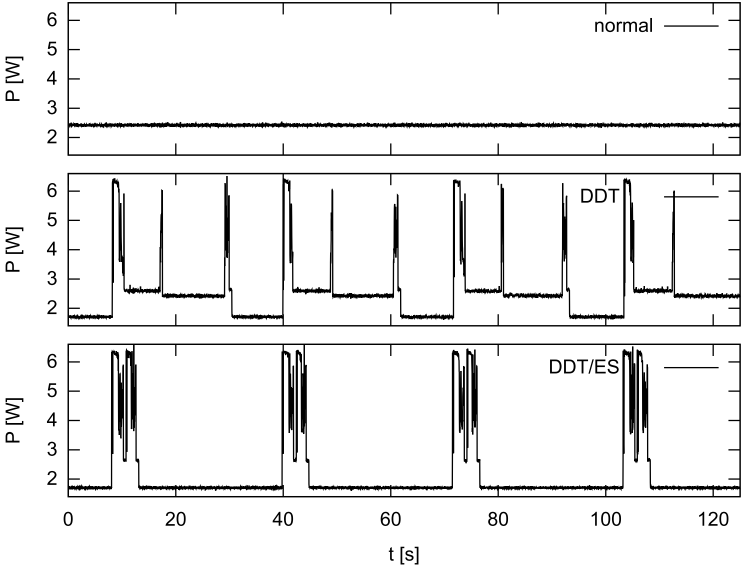
measured break-even period: 22 seconds

audio players

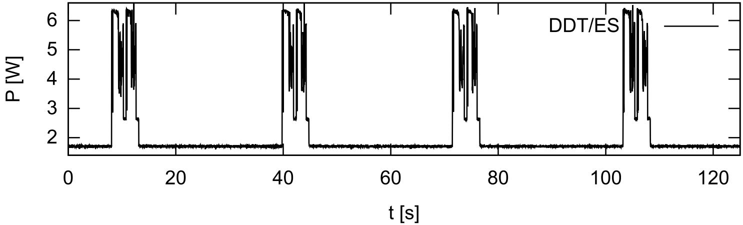
- tested players read from disk too often
- script to simulate player we wanted: (70 kb/s)

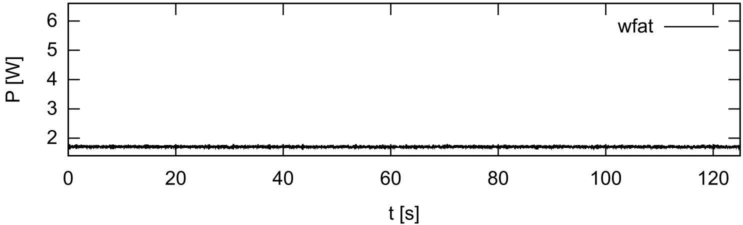
```
#!/bin/sh
echo 3 > /proc/sys/vm/drop_caches
SKIP=0
while [ $SKIP -lt 10000 ]; do
  dd if=/media/SMALL/alina.mp3 of=/dev/null \
     skip=$SKIP count=512
  SKIP=$[$SKIP+512]
  sleep 30
done
```

measurement results



measurement results





demo