Persistent log with UBI

Matthieu CASTET - www.parrot.com

25 September 2013

Parrot

Goal

- Log must be persistent on product : Nand flash
  - Can be used to trace system updates
- Has to be independent/hidden of application filesystem
  - not on / filesystem
- Has to be usable on shipped products
  - difficult to resize all partitions

Which interface to use ?

- Raw flash (MTD)
- UBI
- Nand filesystem (UBIFS, ...)

Parrot

Flash device vs block device 1/2

Block device	Flash device
Consists of sectors	Consists of eraseblocks
Sectors are small (512, 1024 B)	Eraseblocks are larger (typically 128KB)
Maintains 2 main operations: • read sector • write sector	Maintains 3 main operations: • read from eraseblock • write to eraseblock • erase eraseblock

Flash device vs block device 2/2

Block device	Flash device
Bad sectors are re-mapped and hidden by hardware (at least in modern LBA hard drives)	Bad eraseblocks are not hidden and should be dealt with in soft-ware
Sectors are devoid of the wear-out property	Eraseblocks wear-out and become bad and unusable after about 10^3 (for MLC NAND) - 10^5 (NOR, SLC NAND) erase cycles

- Flash device is more difficult to handle (ecc, bad block, eraseblock, ...)

MTD

- MTD stands for "Memory Technology Devices"
- Provides an abstraction of flash devices
- Hides many aspects specific to particular flash technologies
- Provides uniform API to access various types of flashes
- E.g., MTD supports NAND, NOR, ECC-ed NOR, DataFlash, OneNAND, etc
- Provides partitioning capabilities

UBI

- Stands for "Unsorted Block Images"
- Provides an abstraction of "UBI volume"
- Has kernel API and user space API (/dev/ubi0)
- Provides wear-leveling
- Hides bad eraseblocks
- Allows run-time volume creation, deletion, and re-size
- Is somewhat similar to LVM, but for MTD devices

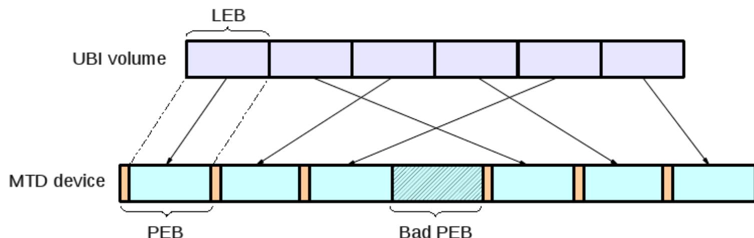
UBI volume vs MTD device

MTD device	UBI device
Consists of physical eraseblocks (PEB), typically 128 KB	Consists of logical eraseblocks (LEB), slightly smaller than PEB (e.g 126/124 KB)
Has 3 main operations • read from PEB • write to PEB • erase PEB	Has 3 main operations • read from LEB • write to LEB • erase LEB
May have bad PEBs	Does not have bad LEB (handle a certain amount of bad PEB)
PEBs wear out	LEBs do not wear out - UBI spread the I/O load across the whole flash
MTD devices are static	UBI volumes are dynamic

Parrot

Main idea behind UBI

- Maps LEBs to PEBs
- Any LEB may be mapped to any PEB
- Eraseblock headers store mapping information and erase count



Parrot

Other

- Handle bit-flips by moving data to a different PEB
- Configurable wear-leveling threshold
- Atomic LEB change
- Volume update/rename operation
- Suitable for MLC NAND
- Performs operations in background
- Works on NAND, NOR and other flash types
- Tolerant to power cuts
- Simple and robust design
- easy support in bootloader

UBIFS

- Filesystem on top of UBI (2.6.27 2008-10)
- Needs a minimal number of LEBs to work : 17
  - http://www.linux-mtd.infradead.org/faq/ubifs.html#L_few_lebs
- Complex filesystem : few (rare) corruptions seen on products

Parrot

log over UBI

- we have a UBI device (for linux kernel) with free space
- UBIFS has too much overhead

Parrot

UBI user API

- `include/mtd/ubi-user.h`
  - device attach
  - device detach
  - volume create
  - volume delete
  - volume resize
  - volume rename

  - volume update (static volume)
  - LEB erase
  - LEB atomic change
  - LEB map
  - LEB unmap

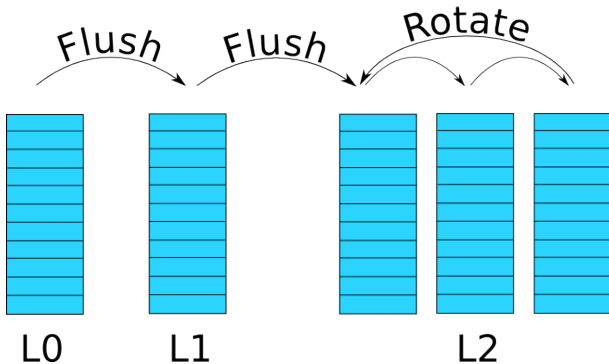
  - lseek
  - read
  - write

Parrot

ulog

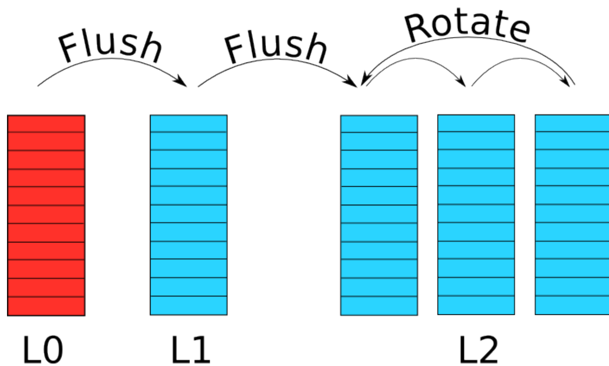
- A log entry is much smaller than a page size (512B-4KB)
- A cache is needed (in flash)
- ulog uses a dynamic volume to have per LEB write

ulog



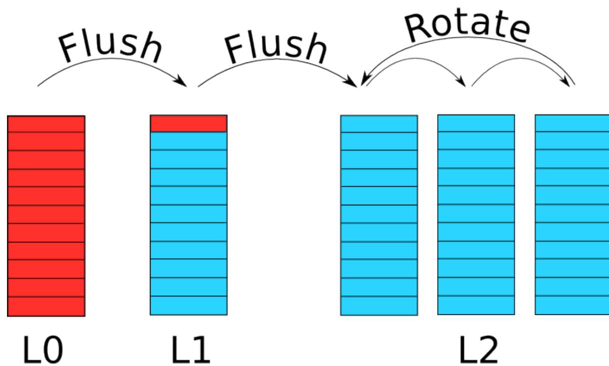
Parrot

ulog : flush 1



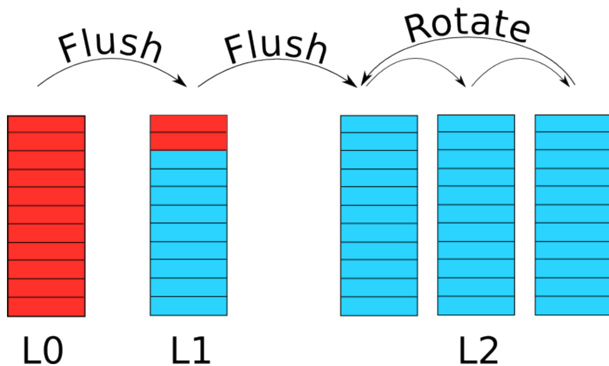
Parrot

ulog : flush 2



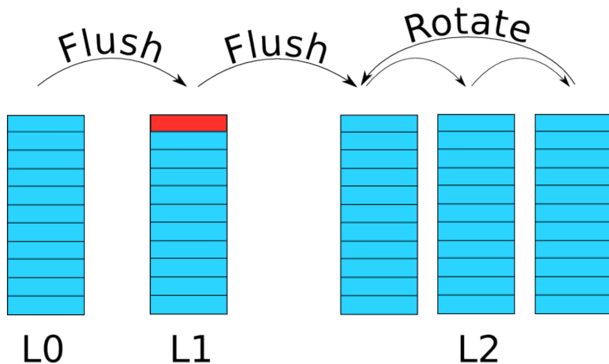
Parrot

ulog : flush 2



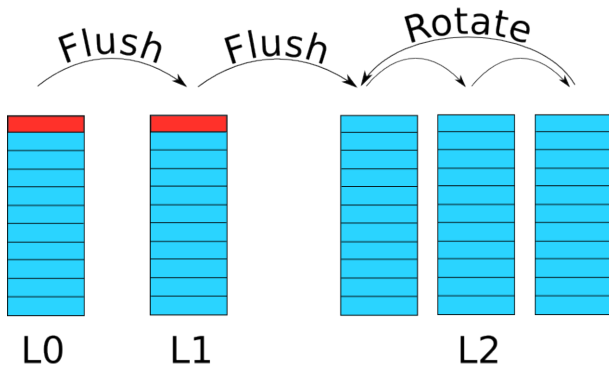
Parrot

ulog : flush 3



Parrot

ulog : flush 4



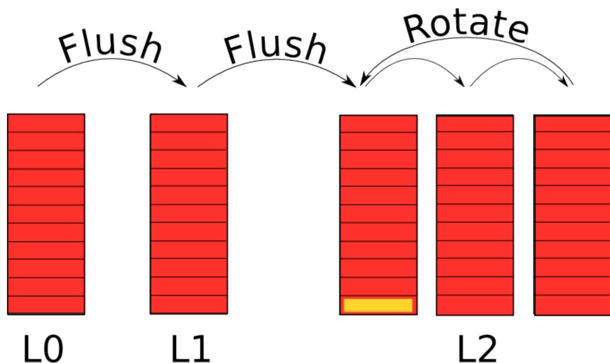
Parrot

Data flush

- Copy data from level N to level N+1
- Merge pages to try to fill pages from level N+1
  - if level 0 has 64 entries of about 20 B, it can be merged in a page of 2 KB.
- Can be recursive !
  - flush of L0, but L1 is full
  - flush of L1 needed
  - ...

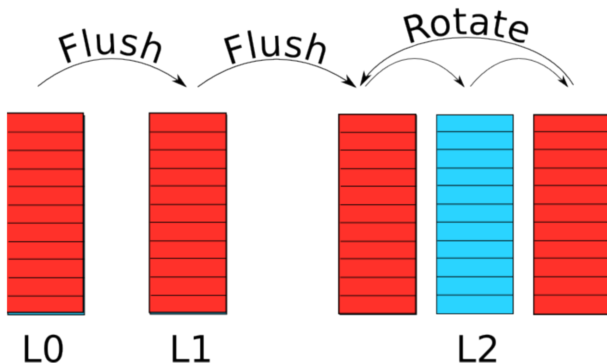
Parrot

ulog : rotate 1

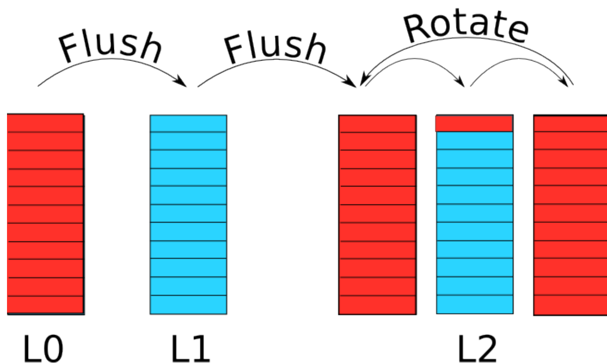


Parrot

ulog : rotate 2

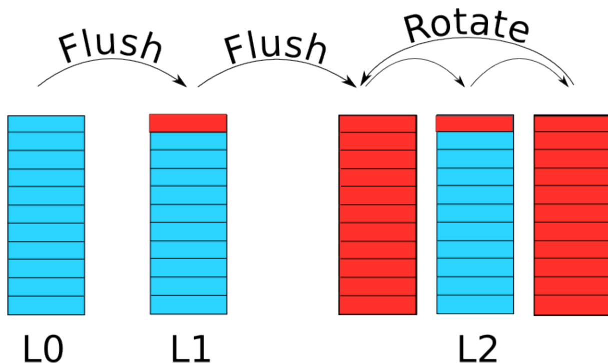


ulog : rotate 3



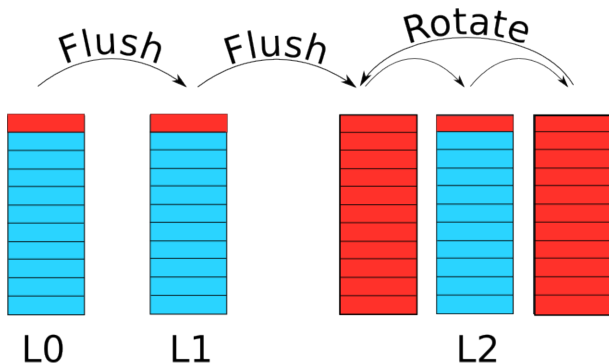
Parrot

ulog : rotate 4



Parrot

ulog : rotate 5



Parrot

Data rotate

- Clean next LEB (if not empty)
- Write data on it

Parrot®

Data format

- Header (32 bits)
  - size (24 bits)
  - version (3 bits) (relevant for first page)
- Log data

Scan

- Parse header for all LEBs
  - LEB version
  - LEB index (which pages have data)
- For last level (L2), find current LEB
  - All empty use first (L2) LEB
  - Use LEB version

Parrot

Scan

- Parse header for all LEBs
  - LEB version
  - LEB index (which pages have data)
- For last level (L2), find current LEB
  - All empty use first (L2) LEB
  - Use LEB version

Parrot

ulog API

- `ulog_init`
- `ulog_destroy`
- `ulog_read`
- `ulog_printf`
- `ulog_vprintf`
- `ulog_flush`

ulog demo

Parrot

TODO

- any remark?

Parrot

TODO

- any remark?
- Need to correctly handle all power failure cases during a flush
  - Can be complex in case of cascaded flush
- Some data are currently duplicated

Questions ?

- Merci pour votre attention !
- Thanks for your attention!
- Questions?