



Evaluation of UBI and UBIFS

TOSHIBA CORPORATION
Core Technology Center
Embedded System Technology Development Dept.
UWATOKO Katsuki
Oct 2, 2009

Agenda

- **Background**
- **Topic**
 - 1. Boot time**
 - 2. Flash space overhead**
 - 3. UBI scrubbing on preempt kernel**
 - 4. Sub-page write verify**
 - 5. Error handling**
- **Summary**
- **References**

Background

- Large size NAND flash memory is commonly used in embedded systems & Flash File systems for Larger size NAND are necessary.

- **UBIFS : Unsorted Block Image File System**
(Artem Bityutskiy, Adrian Hunter)

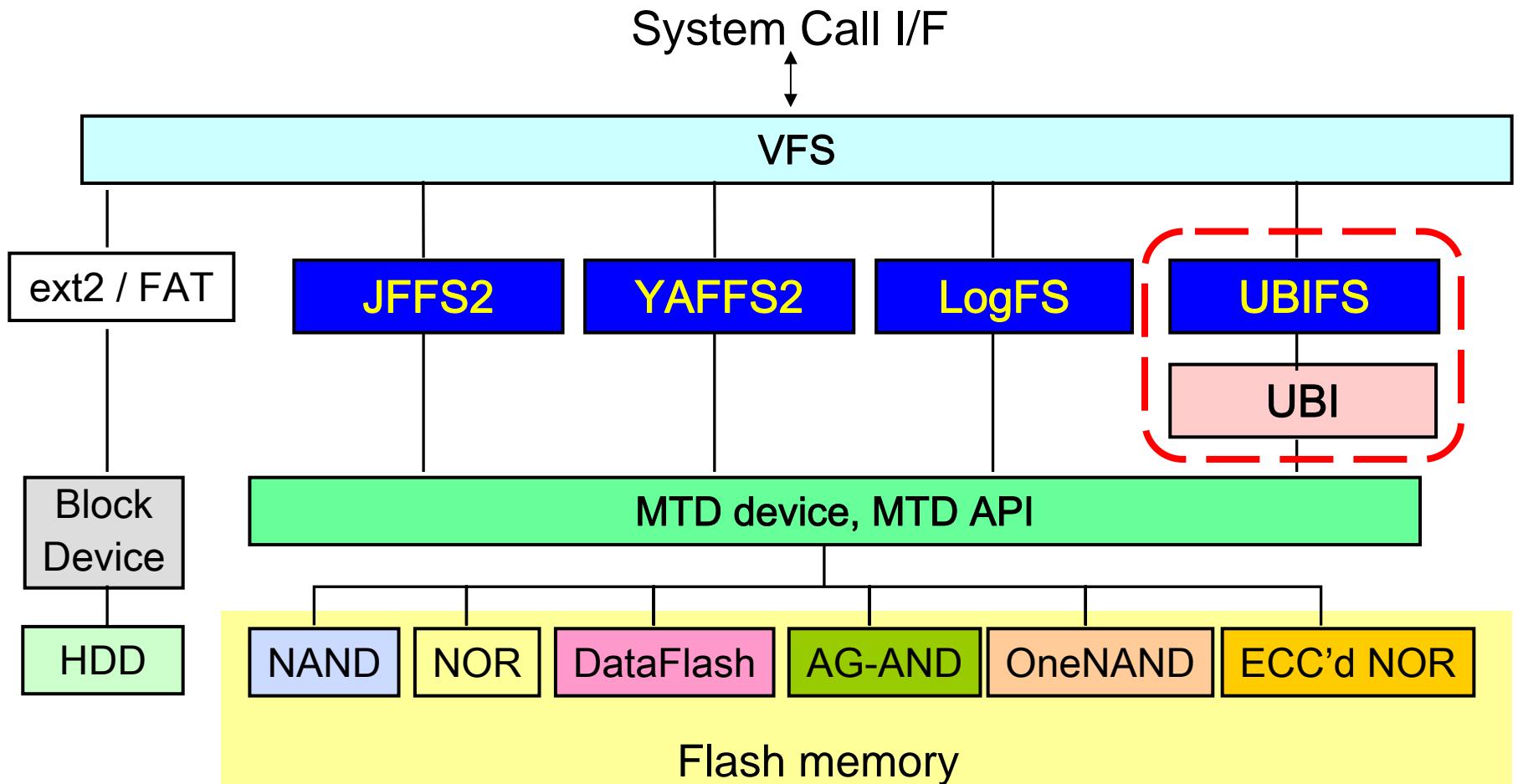
- Mainlined in 2.6.22 in Jul 2007.
- Works on top of UBI volumes.
- Write-back, Compression, Journal, ...

- **UBI : Unsorted Block Images** (Artem Bityutskiy)
 - Mainlined in 2.6.27 in Oct 2008.
 - Works on top of MTD partition.
 - Logical Volume, Global Wear-leveling, Scrubbing, Journal, ...



UBI/UBIFS is one of the best file system. But one would run into some issues when evaluating it. This reports those issues.

Background



VFS: Virtual File System
MTD: Memory Technology Device

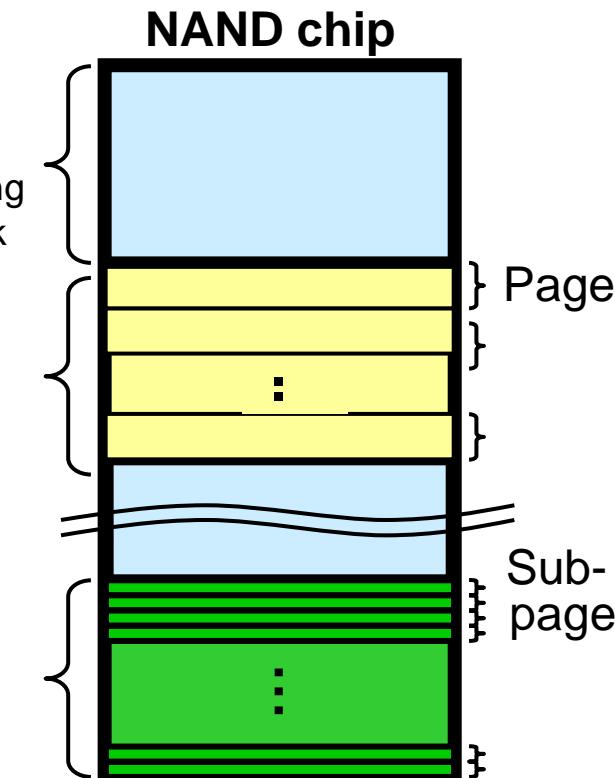
Background

- **Software**
 - Linux kernel : Vanilla kernel 2.6.20.19 + Original patch for embedded systems
 - UBIFS / UBI : Vanilla kernel 2.6.30
 - MTD : Vanilla kernel 2.6.20.19
 - MTD NAND driver : txx9ndfmc (Original patch for the product's CPU)

• Hardware

- Board : Digital product development board
- CPU : MIPS 528 MHz (I\$/D\$: 64KB/64KB)
- RAM(kernel) : 256 MB (64 MB)
- NAND :

	Toshiba	Hynix
Bus	8 bit	8 bit
Total Size	Data	32 MB
	OOB	1 MB
Erasing block	16 KB	16 KB
Page	512 bytes	512 bytes
Sub-page	256 byte	256 byte

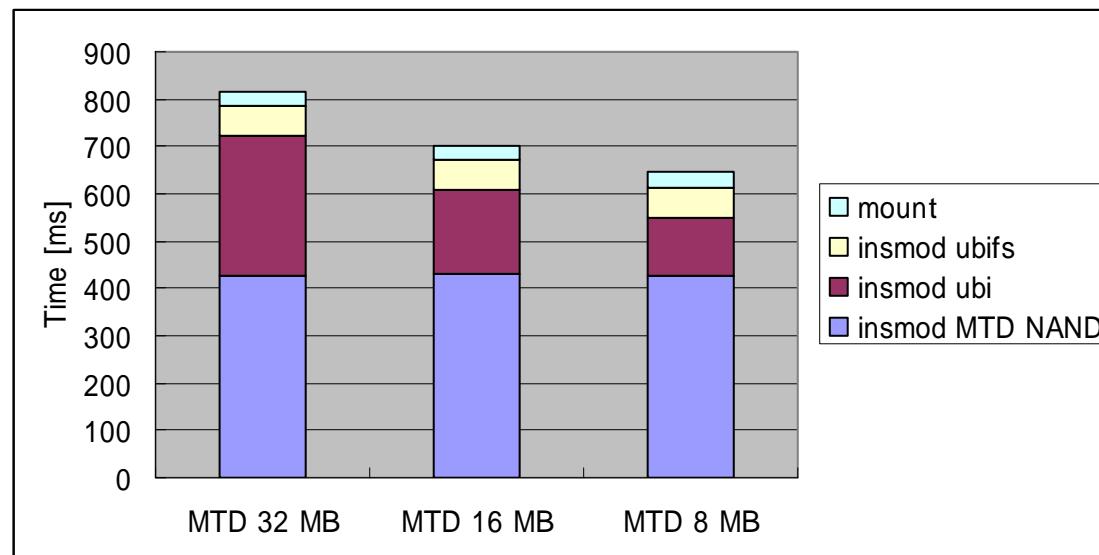


Topic 1 : Boot time

The correlation between the time and the size

MTD 32MB partition, UBI 1 volume (Toshiba NAND)

Command	MTD 32 MB	MTD 16 MB	MTD 8 MB
insmod MTD NAND	428	429	428
insmod ubi	295	181	123
insmod ubifs	61	62	62
mount	30	30	32



Topic 1 : Boot time

The MTD NAND driver initialization is slow.

**The time is not depend on the size of MTD partition,
is depend on the size of NAND chip.**

- **Cause**

The building BBT (Bad Block Table) is slow.

- Reading 1 OOB (16 Byte) per a EraseBlock
- BBT of whole chips (MTD0) is build in the MTD NAND driver initialize.

- **Ideas of an improvement**

- Building BBT per MTD partitions.
- Using “BBT on Flash” - BBT is in a separate internal volume.
- Using Large Page Size NAND

Topic 1 : Boot time

The UBI initialization is slow.

The time is depend on the size of MTD partition.

- Cause

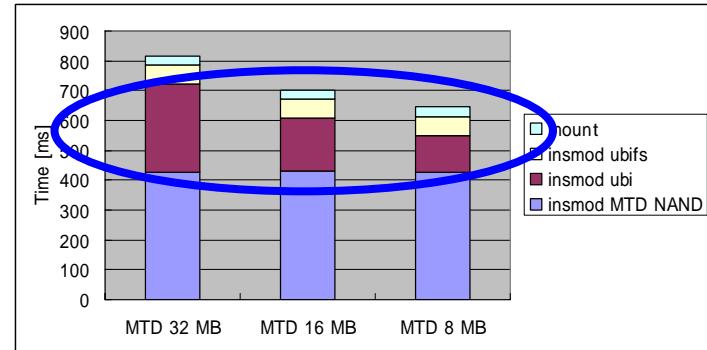
The reading of EC (Erase Counter) and VID (Volume Identifier) is slow.

- Reading 1 page (512 Byte) per a EraseBlock (2048 PEB * 512 = 1 MB)
- BBT of whole chips (MTD0) is build in the MTD NAND driver initialize.
- http://www.linux-mtd.infradead.org/doc/ubi.html#L_scalability

“Scalability issues”

- The ideas of an improvement

- Whole EC/VID are in a separate internal volume like “BBT on Flash”
- http://www.linux-mtd.infradead.org/faq/ubi.html#L_attach_faster
“How do I speed up UBI initialization ”



Topic 2 : Flash space overhead

- **Problem**

“df” command reports too few free space.

The result of df on 32 MB (32768 1k-blocks) MTD partition...

Filesystem	1k-blocks	Used	Available	Use%	Mounted on
ubi0_0	22960	20	21468	0%	/mnt/omote

- **Factors**

“http://www.linux-mtd.infradead.org/faq/ubifs.html#L_df_report” describes the details.

“Why df reports too few free space?”

- compression
- write-back

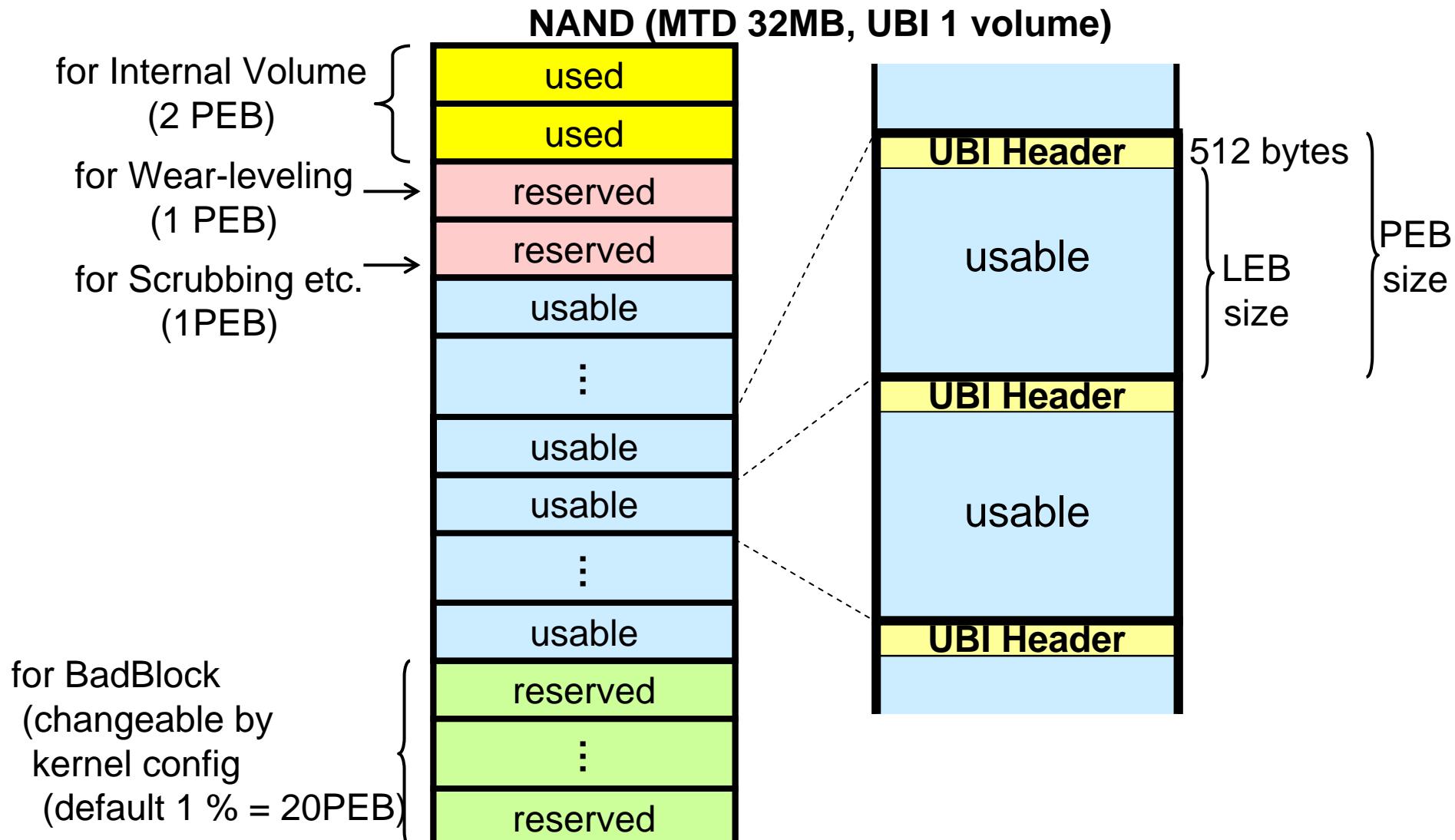
The accurate size is unknown because the compression ratio depends on each file.

- space wastage at the end of logical eraseblock
 - as more fully discussed herein after
- garbage-collection

The accurate result of garbage-collection is not predictable.

Topic 2 : Flash space overhead

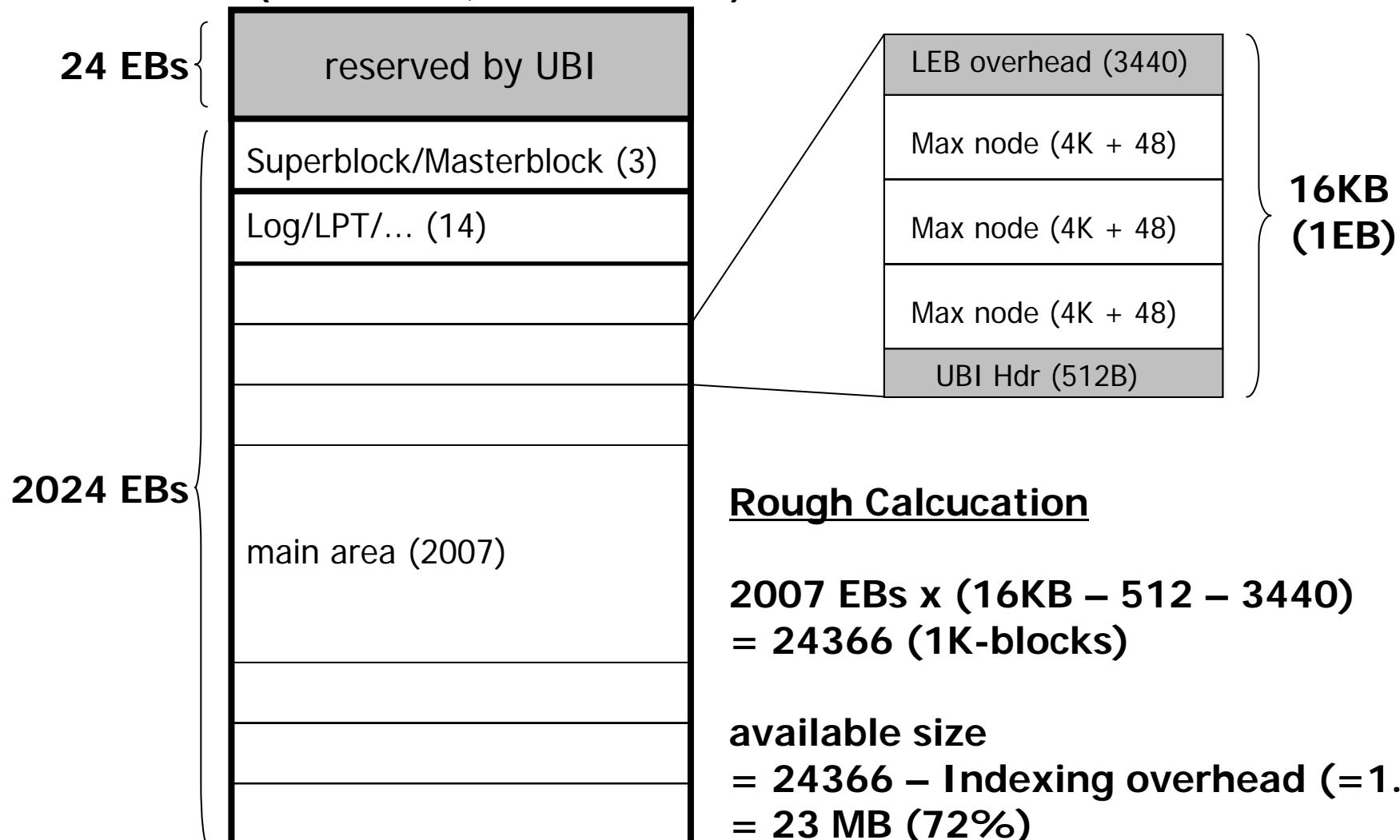
- The factor of UBI



Topic 2 : Flash space overhead

- The factor of UBIFS

NAND (MTD 32MB, UBI 1 volume)



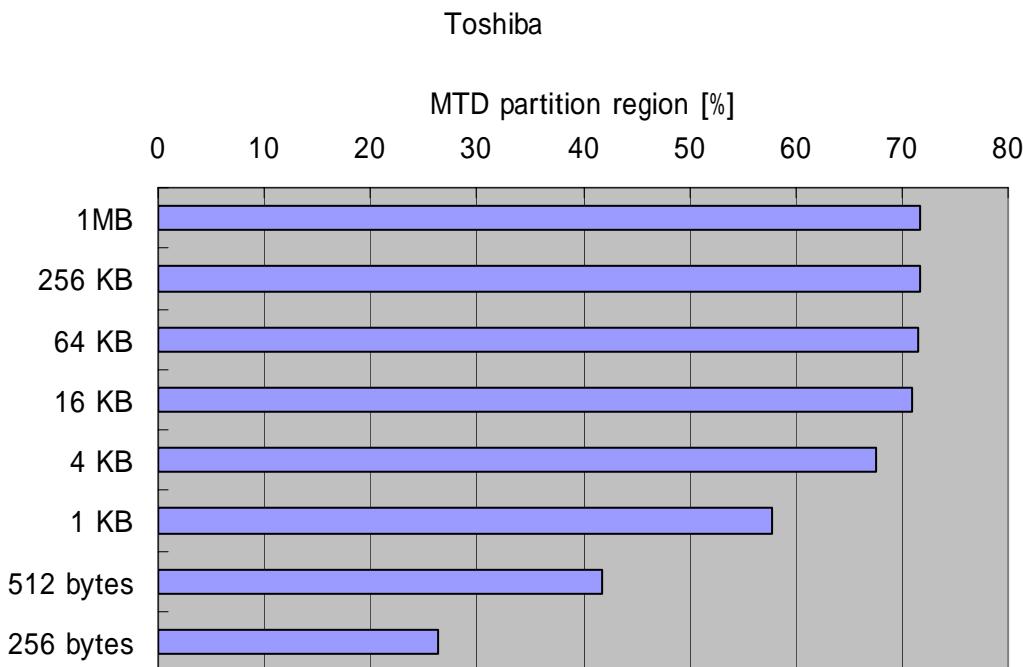
Topic 2 : Flash space overhead

- The actual total size of user data

Conditions:

- MTD partition size: 32MB, UBI volume number: 1
- LZO compression enable
- Random Data Random Data (The compression rate is low)
- “df” command reports “available 22960 KB” when empty.

file size	writable data [MB] (%)
1MB	23.5 (71.8)
256 KB	23.5 (71.8)
64 KB	23.4 (71.6)
16 KB	23.2 (70.9)
4 KB	22.1 (67.5)
1 KB	18.9 (57.8)
512 bytes	13.6 (41.7)
256 bytes	8.6 (26.3)



	Description	# of blocks
UBI (2048 EBs)	For Internal-Vol, Wearleving, Scrubbing	4
	For Bad Block	20
UBIFS (2024 EBs)	Super Block/Master Block	3
	LOG/LPT..	14
	Main Area (2007 EBs)	24
	Empty Block	19
	Index Block	1964
	Data Block	

- **Typical Data Block**

LEB 893 lprops: free 3072, dirty 368 (used 12432), flags 0x0

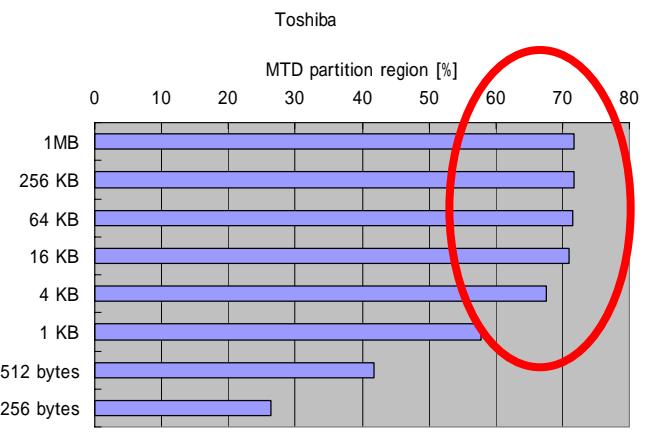
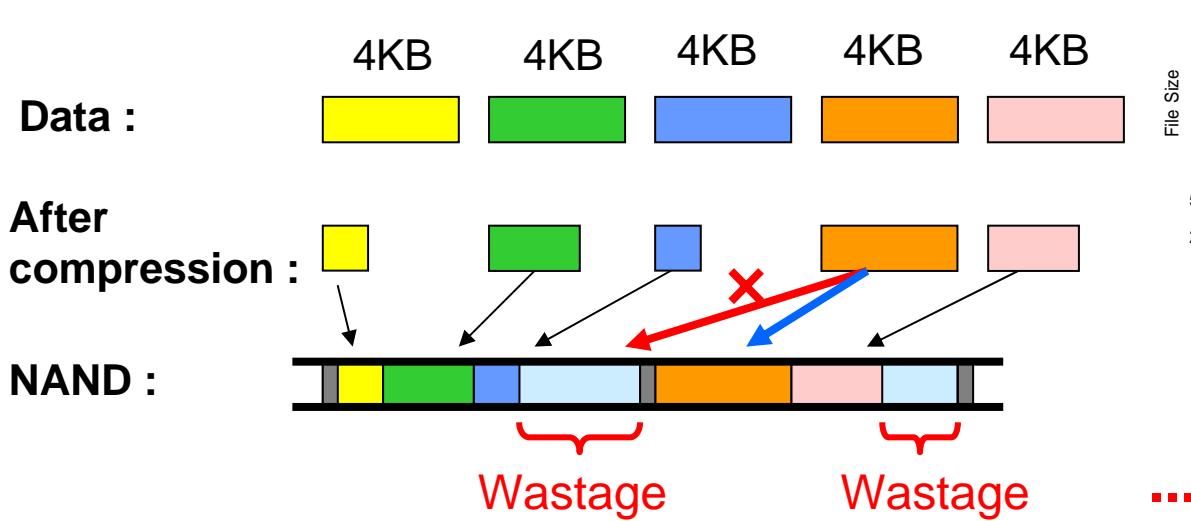
Topic 2 : Flash space overhead

- Cause

Free space at the tail of PEB

http://www.linux-mtd.infradead.org/doc/ubifs.html#L_space

“Flash space accounting issues” – “Wastage”



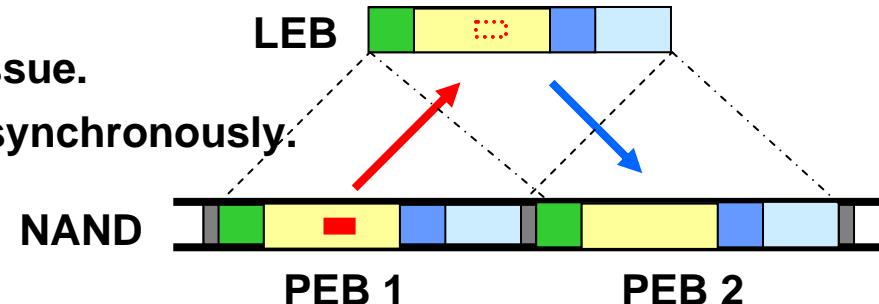
The ideas of an improvement

- change the unit size (`UBIFS_BLOCK_SIZE`) of UBIFS to smaller (must be power of two)
 - influence to the compression ratio and the performance
- use Large Page size NAND

Topic 3 : UBI scrubbing on preempt kernel

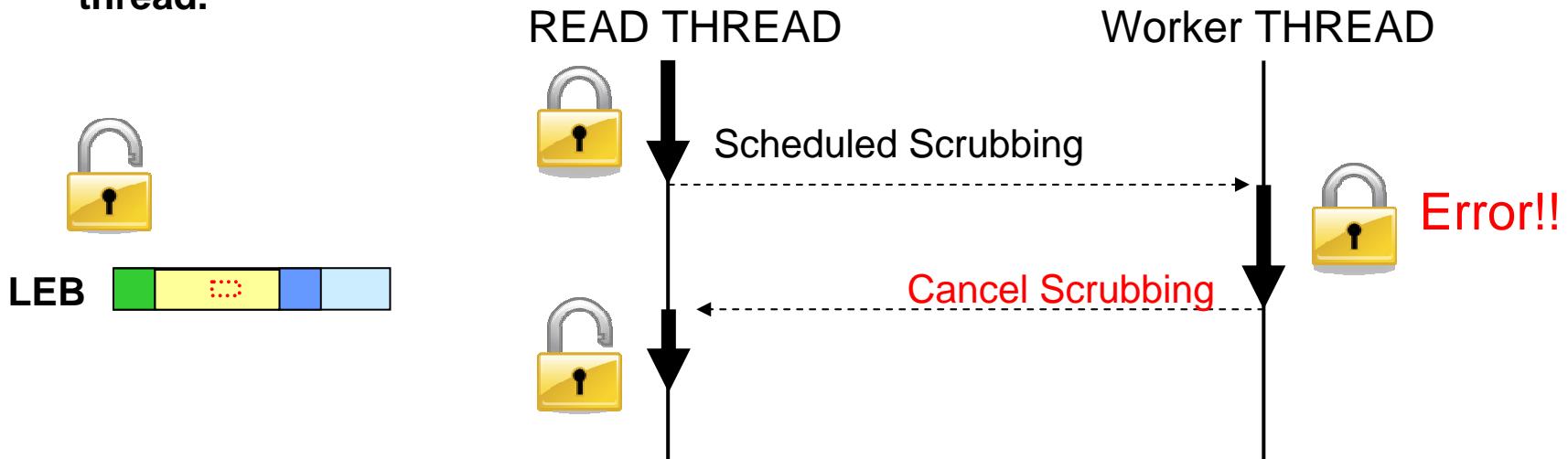
- **UBI “Scrubbing” function**

- When a bit-flip (ECC 1 bit error) is detected in a physical erase block, the block is move to other physical erase block.
- It is beneficial for the “read disturb” issue.
- Worker thread execute “scrubbing” asynchronously.



- **Symptom**

- In the case of Preempt Kernel, the scrubbing could be canceled. Because the read thread holding the lock of LEB could be preempted by the wearleveing thread.



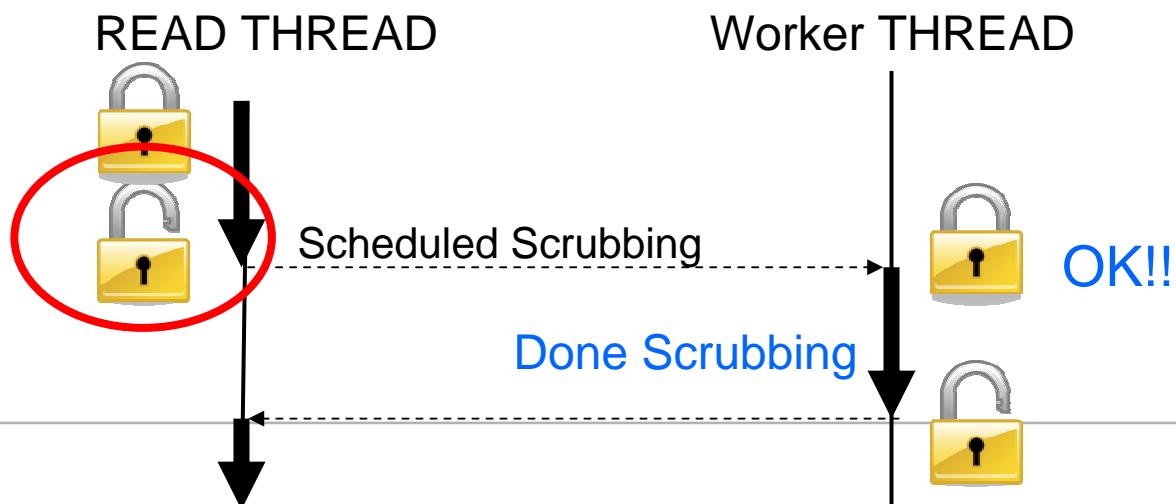
Topic 3 : UBI scrubbing on preempt kernel

- Patch
 - The lock of LEB is released before queuing a scrubbing.
 - It is not necessary to hold the lock when queuing it.
- [drivers/mtd/ubi/eba.c]

```
int ubi_eba_read_leb(struct ubi_device *ubi, struct ubi_volume *vol, int Inum,
                      void *buf, int offset, int len, int check)
{
    (中略)
+        leb_read_unlock(ubi, vol_id, Inum);

    if (scrub)
        err = ubi_wl_scrub_peb(ubi, pnum);

-
-        leb_read_unlock(ubi, vol_id, Inum);
    return err;
}
```



Topic 4 : Sub-page write verify

- **MTD NAND “Verify NAND page writes” function**
 - is the write verify function per a NAND physical page.
 - is implemented in MTD NAND driver.
- **Symptom**
 - does not support per a sub page.
the another sub page in the same page would not be managed properly.
 - UBI uses a sub page write in writing VID header.
 - http://www.linux-mtd.infradead.org/faq/ubi.html#L_subpage_verify_fail
“I get “ubi_io_write: error -5 while writing 512 bytes to PEB 5:512”

If you have a 2048 bytes per NAND page device, and have CONFIG_MTD_NAND_VERIFY_WRITE enabled in your kernel, you will need to turn it off. The code does not currently (as of 2.6.26) perform verification of sub-page writes correctly. As UBI is one of the few users of sub-page writes, not much else seems to be affected by this bug.

Not only “a 2048 bytes per NAND page device”.
(ex. This NAND. Page: 512 bytes, Sub-page: 256 bytes)

Topic 4 : Sub-page write verify

- **Patch**

The function for write verifying with column (offset) and size is added to MTD NAND structure in MTD NAND driver.

[include/linux/mtd/nand.h]

```
struct nand_chip {  
    void __iomem      *IO_ADDR_R;  
    void __iomem      *IO_ADDR_W;  
  
    uint8_t           (*read_byte)(struct mtd_info *mtd);  
    u16              (*read_word)(struct mtd_info *mtd);  
    void             (*write_buf)(struct mtd_info *mtd, const uint8_t *buf, int len);  
    void             (*read_buf)(struct mtd_info *mtd, uint8_t *buf, int len);  
    int              (*verify_buf)(struct mtd_info *mtd, const uint8_t *buf, int len);  
+ #ifdef CONFIG_MTD_NAND_VERIFY_WRITE_SUBPAGE  
+     int            (*verify_buf_column)(struct mtd_info *mtd, const uint8_t *buf, int len,  
+                                         int column, int verify_size);  
+  
+ #endif  
    void             (*select_chip)(struct mtd_info *mtd, int chip);  
    int              (*block_bad)(struct mtd_info *mtd, loff_t ofs, int getchip);  
    int              (*block_markbad)(struct mtd_info *mtd, loff_t ofs);
```

Topic 4 : Sub-page write verify

[drivers/mtd/mtd/nand/nand_base.c]

```
static int nand_do_write_ops(struct mtd_info *mtd, loff_t to,
                           struct mtd_oob_ops *ops)
{
    (中略)
    /* Partial page write ? */
    if (unlikely(column || writelen < (mtd->writesize - 1))) {
        cached = 0;
        bytes = min_t(int, bytes - column, (int) writelen);
        chip->pagebuf = -1;
        memset(chip->buffers->databuf, 0xff, mtd->writesize);
        memcpy(&chip->buffers->databuf[column], buf, bytes);
        wbuf = chip->buffers->databuf;
    }

    if (unlikely(oob))
        oob = nand_fill_oob(chip, oob, ops);

    ret = chip->write_page(mtd, chip, wbuf, page, cached,
                           (ops->mode == MTD_OOB_RAW));
    if (ret)
        break;

+ #if defined(CONFIG_MTD_NAND_VERIFY_WRITE_SUBPAGE) && !defined(CONFIG_MTD_NAND_VERIFY_WRITE)
+         /* Send command to read back the data */
+         chip->cmdfunc(mtd, NAND_CMD_READ0, 0, page);
+
+         if (chip->verify_buf_column(mtd, wbuf, mtd->writesize, column, bytes))
+             return -EIO;
+
+ #endif
```

Topic 5 : UBIFS/UBI error handling

- **UBIFS became read-only when an error occurs.**
 - for protect data from any further corruption
 - http://www.linux-mtd.infradead.org/faq/ubifs.html#L_sudden_ro
“UBIFS suddenly became read-only – what is this?”
- **Problem**
 - It is possible that NAND accesses (ex. read/write/erase..) will occur.
 - There could be no way to recover from R/O status after shipping.
- **Current Status**
 - UBI: do not switch to R/O mode on read errors
<http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commitdiff;h=b86a2c56e512f46d140a4bcb4e35e8a7d4a99a4b>

Summary

- **Boot Time**
 - The initializations time of MTD NAND driver and UBI depends on the size.
 - The optimization of building BBT in MTD driver and reading UBI information per erase block in UBI is effective for boot time.
ex. “BBT on Flash”, “Large Page Size NAND”..
- **Flash Space Overhead**
 - The unit size (UBIFS_BLOCK_SIZE) is large for small page NAND (16KB).
 - Using Large Page Size NAND
 - change the size to smaller
- **We are preparing the patches in this to the community.**

References

- **MTD, JFFS2, UBIFS, UBI**
<http://www.linux-mtd.infradead.org/>
- **YAFFS2**
<http://www.yaffs.net/>
- **LogFS**
<http://www.logfs.com/logfs/>
- **CE Linux Forum presentations**
 - Yutaka Araki, “Flash File system, current development status”
http://www.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree20?action=AttachFile&do=view&target=celf_flashfs.pdf
 - Katsuki Uwatoko, “The comparison of Flash File system performance”
http://www.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree19?action=AttachFile&do=view&target=celf_flash2.pdf
 - Keijiro Yano, “JFFS2 / YAFFS”
http://www.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree17?action=AttachFile&do=view&target=celf_flashfs.pdf
- **ELC 2009**
 - Toru Homma , “Evaluation of Flash File Systems for Large NAND Flash Memory”
<http://www.celinuxforum.org/CelfPubWiki/ELC2009Presentations?action=AttachFile&do=view&target=ELC2009-FlashFS-Toshiba.pdf>

TOSHIBA
Leading Innovation >>>