Application of UBIFS for Embedded Linux Products

Jaeyoon Jung (jaeyoon.jung@lge.com)

LG Electronics

September 28th

LinuxCon Japan 2010

Agenda

- **Introduction**
- **NAND Flash device**
  - Characteristics of NAND flash
  - MTD device vs. FTL device
  - File systems for NAND flash
- **UBI and UBIFS**
  - Overview of UBI and UBIFS
  - Strengths from the embedded systems’ point of view
- **Mounting time of UBIFS**
  - Experimental results
- **Summary**

Introduction

- **NAND flash memory is one of the most common storage devices in the present embedded systems.**
  - Low price, large capacity, good performance
  - Some inconvenient characteristics to be managed

- **UBIFS is the one of the best file systems for NAND flash.**
  - I/O performance, tolerance to power-failure, wear-leveling
  - It benefits from UBI which provides the better environment for NAND flash.

- **Mounting time is one of the most important parts.**
  - It influences on the boot time of the embedded system.
  - It can depend on the device size, the file system usage and the condition of previous un-mounting.

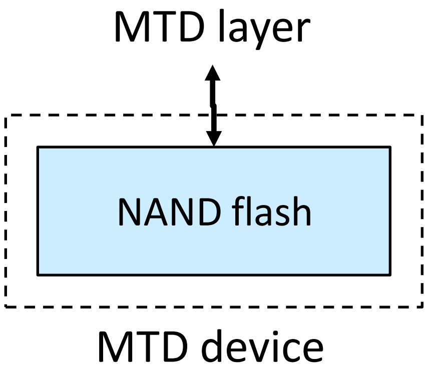
NAND Flash Device: Characteristics of NAND flash

	Characteristic	**Solution**
Bad blocks	NAND flash admits having bad blocks.	Manage bad blocks to avoid using them.
Bit-flips	Data can be disturbed by the influence of adjacent cells being read(read disturb) or programmed (program disturb).	Use ECC to correct disturbed data. Refresh the block to reduce the further bit-flips.
Endurance	All blocks have a limited number of erase/program cycles.	Do the wear-leveling to distribute the number of erasures evenly.

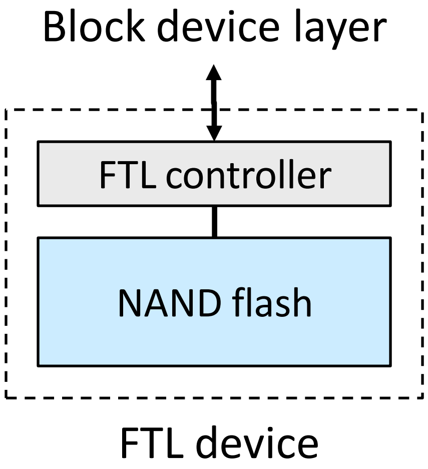
- **For MLC NAND flash**
  - The higher level of ECC is required due to its higher degree of bit-flip.
  - The wear-leveling is more essential because it has a shorter erase/program cycle than the SLC type.

NAND Flash Device: MTD device vs. FTL device (1/2)

- **MTD device**
  - Common interface for flash devices in Linux
  - Provides the abstraction layer for different types of bare flash devices.



- **FTL device**
  - Translation layer for flash devices
  - Requires a controller which emulates the block device interface on top of the flash.
  - Example
    - USB flash drive, SD, MMC, eMMC, SSD, …



NAND Flash Device: MTD device vs. FTL device (2/2)

	MTD (NAND)	FTL
Interface	Direct interface to NAND controller (SoC)	Block device interface (USB mass storage, SD, …)
NAND flash compatibility	Dependent on NAND controller (SoC)	Dependent on FTL controller
Bad blocks, Bit-flips, Endurance	Not managed →Upper layer must manage them.	Managed by FTL controller →Upper layer does not have to manage them.
File systems	JFFS2, YAFFS2, UBIFS, …	FAT, EXT3, EXT4, …

- **Remarks**
  - In case of using FTL device, it is easier to deal with the unexpected change of NAND flash due to discontinuity for example.
  - The cost for FTL device is a little bit higher due to the FTL controller.
  - FTL software is vendor-dependent and the code is not open, so it is hard to verify the problems without help from the vendor.

NAND Flash Device: File systems for NAND flash

- **Linux has many file systems for bare NAND flash devices which have good performance and reliability. Of course, they are open-sourced.**

- **Read-only file systems**
  - Read-only, but fast and safe
    - Suitable for boot partitions
  - Even though they are not designed for NAND flash, they can run on top of MTD with help of an additional layer such as ‘romblock’.
  - Example: SquashFS

- **Read/write file systems**
  - General purpose
  - Supports bad block management, bit-flip handling and wear-leveling.
  - Example: JFFS2, YAFFS2, LogFS, UBIFS

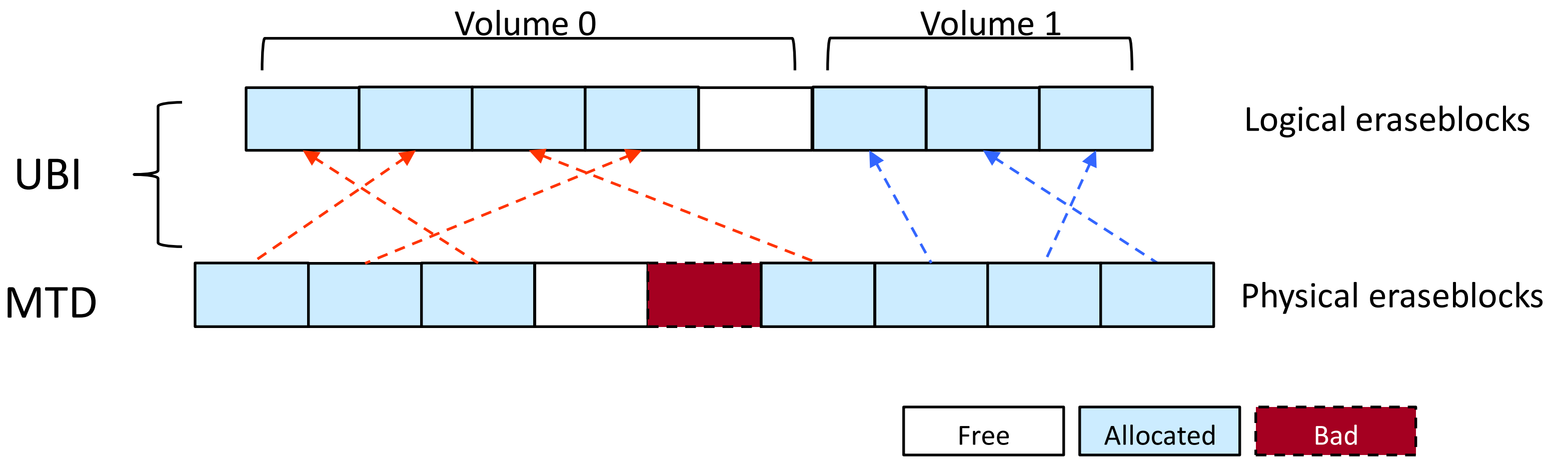
UBI and UBIFS: Overview (1/2)

- **UBI (Unsorted Block Images)**
  - A sub system that provides consecutive logical eraseblocks which are mapped to physical eraseblocks.
  - Volume management system which manages multiple logical volumes on a single raw flash device.
  - Runs on top of MTD.
  - Mainlined since kernel version 2.6.22.

- **UBIFS**
  - A file system for flash which works on top of UBI over MTD.
  - Considered as the next generation of JFFS2.
  - Mainlined since kernel version 2.6.27, 17 July 2008.

UBI and UBIFS: Overview (2/2)

- **Basic concept of UBI**
  - LEB-to-PEB mapping
    - Each PEB has a VID(Volume ID) header which contains a corresponding volume ID and a LEB number.
    - Uses the eraseblock association table in RAM which contains the mapping information.
  - Attaching UBI
    - Gathering the block mapping and volume information by scanning VID headers.
    - It must be done before running a file system on top of UBI.



UBI and UBIFS: Strengths in the embedded systems (1/3)

- **Bad block management (UBI)**
  - Bad PEBs are hidden and managed by UBI.
  - Only non-bad logical eraseblocks are exposed to upper layer.
- **Bit-flip handling (UBI)**
  - Bit-flip can be handled by UBI using ECC.
  - Additional management for possible bad blocks
    - Scrubbing: Moving data from a PEB that has bit-flips to other PEB.
    - Torturing: Doing some tests for a PEB in order to detect it is really bad.
- **Wear-leveling (UBI)**
  - Dynamic wear-leveling
  - Static wear-leveling
  - Global wear-leveling

UBI and UBIFS: Strengths in the embedded systems (2/3)

- **Tolerance to power-failure**
  - Volume table duplication (UBI)
  - Atomic LEB change (UBI)
  - Both UBI and UBIFS can checksum everything when they write data to the flash. (default, optional)

- **On-the-flight compression (UBIFS)**
  - Can compress regular files transparently. (default, optional)
    - Also allows switching the compression on/off on per-inode basis.
    - Supports LZO (faster compression speed) and Zlib (smaller compressed size).

- **High performance (UBIFS)**
  - Write-back, journaling, TNC(tree node cache), LPT(LEB properties tree)
  - Bulk-read (optional)

UBI and UBIFS: Strengths in the embedded systems (3/3)

	JFFS2	YAFFS2	LogFS	UBIFS
Mounting time	Poor	Good	**Excellent**	Good
I/O performance	Good	Good	Fair	**Excellent**
Memory consumption	Fair	**Excellent**	Good	Fair
Wear-leveling	Good	Fair	N/A	**Excellent**
Tolerance to power-failure	Good	Good	Poor	Good

* Reference: "Evaluation of Flash File Systems for Large NAND Flash Memory", Toru Homma, Toshiba

- **Comparison**
  - UBIFS and YAFFS2 show the best result in most cases.
  - LogFS was unstable according to the reference. It was mainlined since 2.6.34 but is still experimental for now.

Mounting time of UBIFS

- **Mounting time**
  - Long mounting time can affect the boot time.
  - Need to evaluate the mounting time of the file system which can be dependent on various conditions.
    - Usage of the file system
    - Size of the device
    - Condition of the previous un-mounting

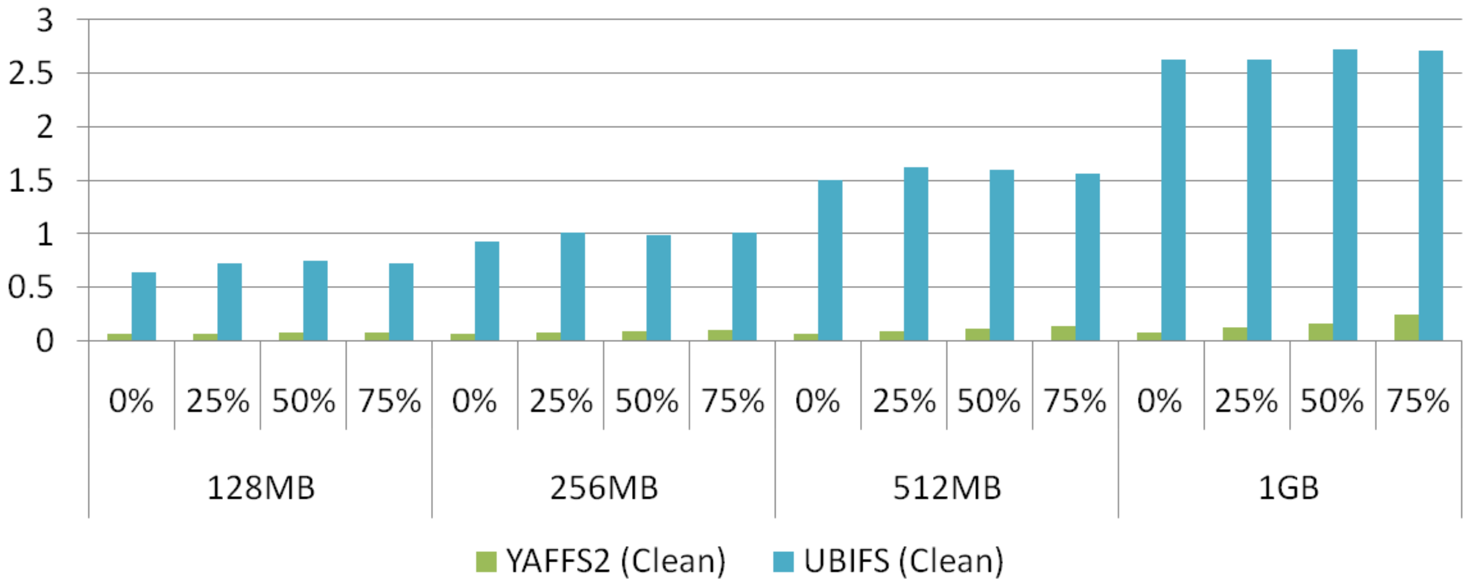
- **Experiment**
  - To evaluate the mounting time of UBIFS for large NAND flash devices under the various conditions.
  - To compare the mounting time of UBIFS with that of YAFFS2.

Mounting time of UBIFS: Experimental Setup

- **Environment**
  - MIPS 400MHz SoC, 512MB DDR2 RAM, Kernel 2.6.31
  - NAND flash
    - 2GB, MLC type, 4bit ECC, 4KB page (512KB eraseblock), no sub-page support
    - Erase: 120.99MB/s, Read: 6.97MB/s, Write: 5.08MB/s (measured practically)
  - UBIFS: Default configuration
    - Mounting time(seconds) = attaching time of UBI + mounting time of UBIFS
  - YAFFS2: updated from CVS on September 6, 2010
    - Mounting time(seconds) = mounting time of YAFFS2

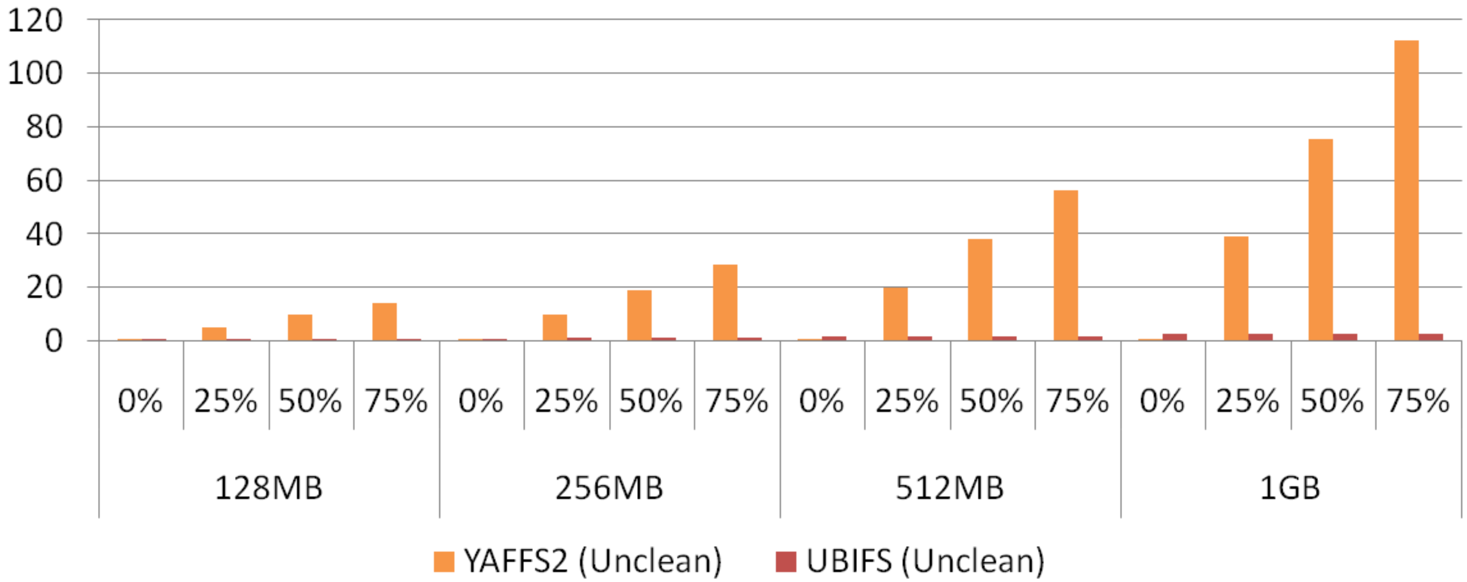
- **Conditions**
  - File system usage: 0%, 25%, 50%, 75% (Filled with 1MB-files)
  - Device size: 128MB, 256MB, 512MB, 1GB
  - Clean reboot: Mounting after the normal reboot
  - Unclean reboot: Mounting after writing 1MB of file and cutting the power source suddenly.

Mounting time of UBIFS: Experimental Result (1/5)



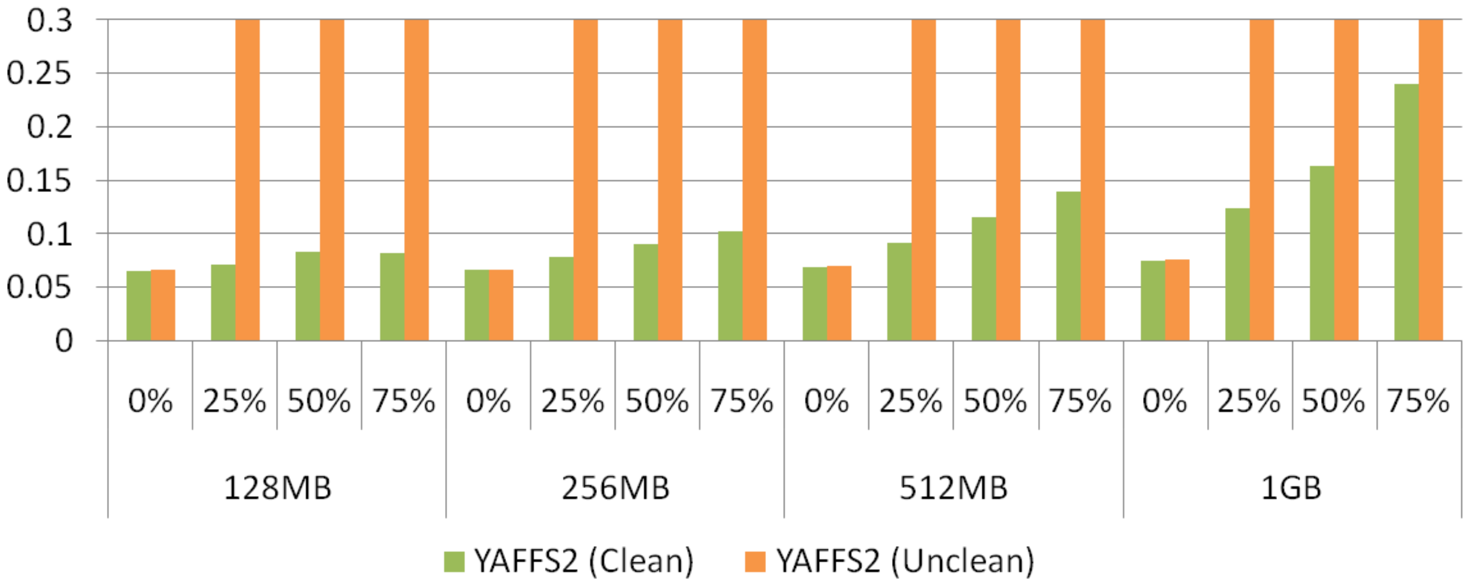
- **After the clean reboot**
  - The mounting time of UBIFS is good but longer than YAFFS2 relatively.
    - UBI has a dependency on the device size because it should scan all VID headers in PEBs when attaching.
    - For huge devices, UBIFS can not be a good choice in terms of mounting time.
  - YAFFS2 mounts very fast after the clean reboot.
    - With help of the checkpoint.

Mounting time of UBIFS: Experimental Result (2/5)



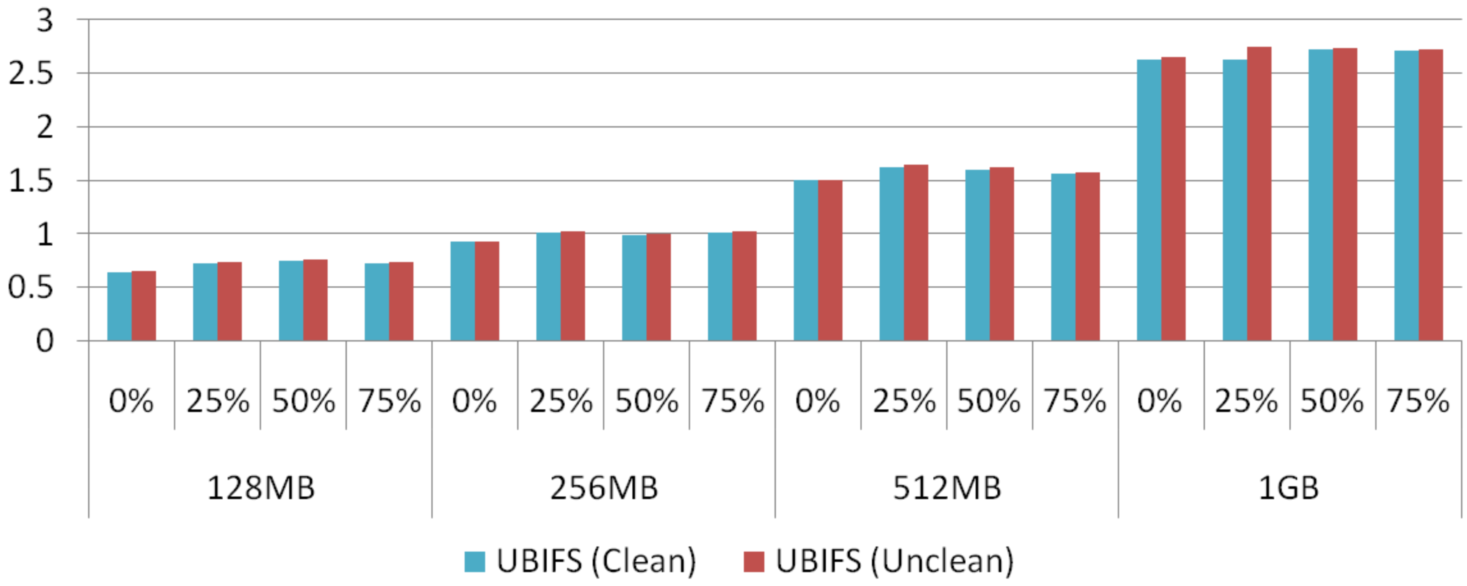
- **After the unclean reboot**
  - The mounting time of UBIFS is similar to the case of the clean reboot.
  - The mounting time of YAFFS2 is dramatically increased.
    - No checkpoint due to sudden power-down.
    - Without the checkpoint it should scan all chunks in use.
    - Mounting time ≈ Time for scanning all chunks

Mounting time of UBIFS: Experimental Result (3/5)



- **YAFFS2: Comparison between clean and unclean reboot**
  - For YAFFS2, the existence of checkpoint makes a huge difference in terms of the mounting time especially for large NAND devices.
  - YAFFS2 is not scalable in both cases with respect to the file system usage and the device size.

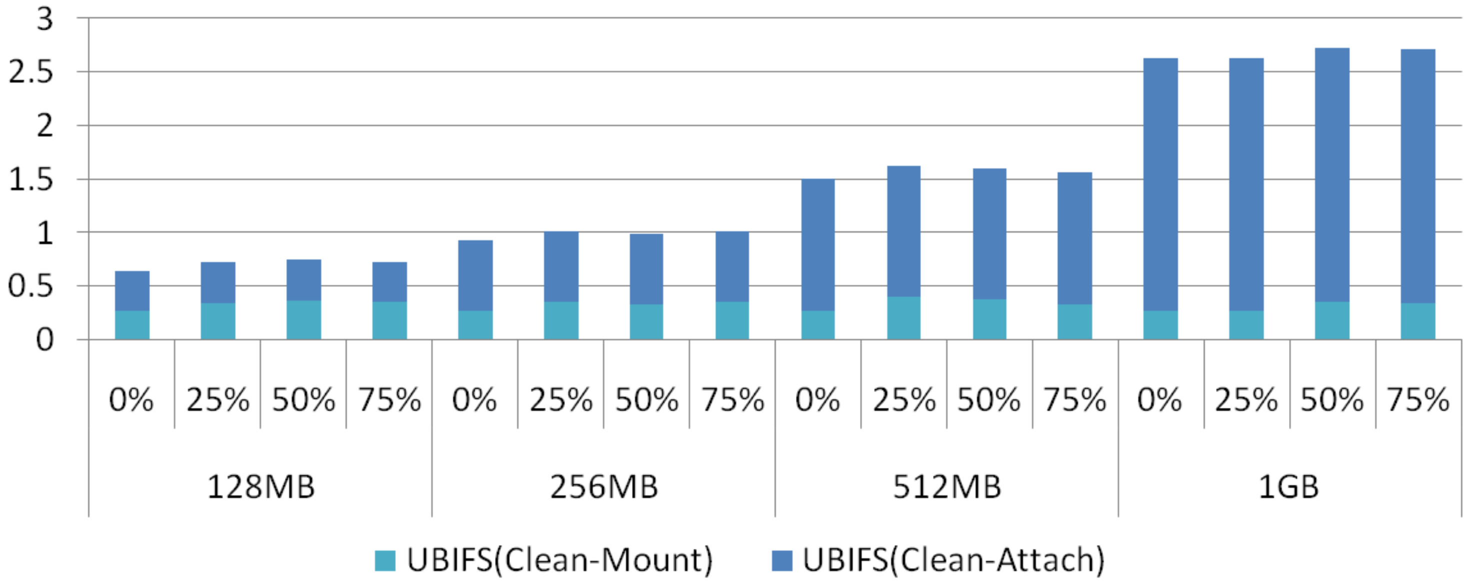
Mounting time of UBIFS: Experimental Result (4/5)



- **UBIFS: Comparison between clean and unclean reboot**
  - Not a big difference between both cases.
    - Mounting time after the unclean reboot takes a bit longer. (avg=0.02s)
    - The difference almost comes from the journal recovery.

LG Electronics

Mounting time of UBIFS: Experimental Result (5/5)



- **Attaching time and mounting time of UBIFS**
  - The larger device size is the longer attaching time of UBI.
    - UBI scales linearly in terms of the device size.
  - The actual mounting time of UBIFS is independent both on the device size and the usage of file system
    - It stores and maintains the index data on the flash.
    - Only the amount of journal to be recovered matters.

Summary

- **NAND flash is commonly used in the embedded Linux products nowadays but it has characteristics which should be managed carefully.**

- **UBIFS is the best file system for the large NAND device from the various aspects such as wear-leveling, tolerance to power failure and I/O performance.**

- **UBIFS mounts fairly fast under the various conditions. It also mounts fast in case of the unclean reboot while YAFFS2 shows a poor performance in the same case.**

- **UBI has a scalability issue with respect to the device size. It affects UBIFS too because UBI must be attached before mounting UBIFS.**

References

- **Websites**
  1) UBI, http://www.linux-mtd.infradead.org/doc/ubi.html
  2) UBIFS, http://www.linux-mtd.infradead.org/doc/ubifs.html

- **Presentation materials**

  1) The Inconvenient Truths of NAND Flash Memory, Jim Cooke, Micron Technology
  2) Evaluation of Flash File Systems for Large NAND Flash Memory, Toru Homma, Toshiba
  3) An examination of UBI, Shinji Namihira, Toshiba
  4) UBI – Unsorted Block Images, Artem Bityutskiy, Nokia
  5) UBIFS file system, Adrian Hunter , Artem Bityutskiy, Nokia