Increasing Compression Performance of Block Based File Systems

Ferenc Havasi

havasi@inf.u-szeged.hu

Szeged, 2004.

Motivation

- Small electronics devices are more and more popular: mobile phones, digital cameras, mp3 players, USB drives and PDAs
- These devices uses mostly flash memory as storage device
- Flash memory is costly ($\rightarrow$ improve compression)

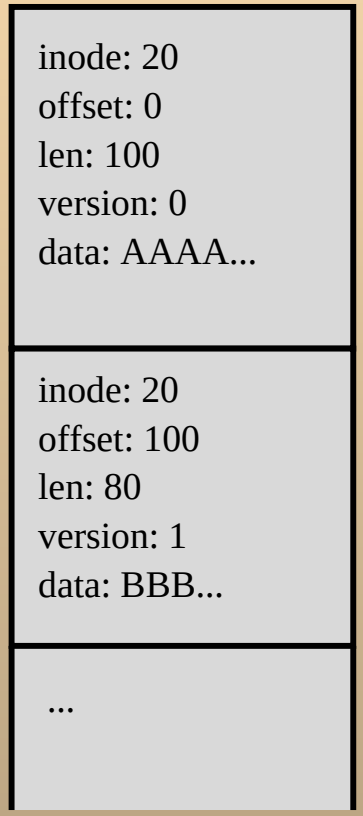
About Flash Memory

- Low power, high density, non-volatile storage. Two kinds: NOR and NAND
- Reading: (almost) the same as reading RAM
- Writing: clearing bits ($1 \rightarrow 0$)
- Bits can be reseted ($0 \rightarrow 1$) only in erase blocks of tipically 128KB
- Limited lifetime - typ. 100,000 erase cycles

Storing Information on Flash

- Emulate standard block device and use an ordinary files system (FAT, EXT2, NTFS, ...) - dangerous
- Use file system designed specially for flash: YAFFS, JFFS, JFFS2 (log-structured file systems)

What does "log-structured" mean?



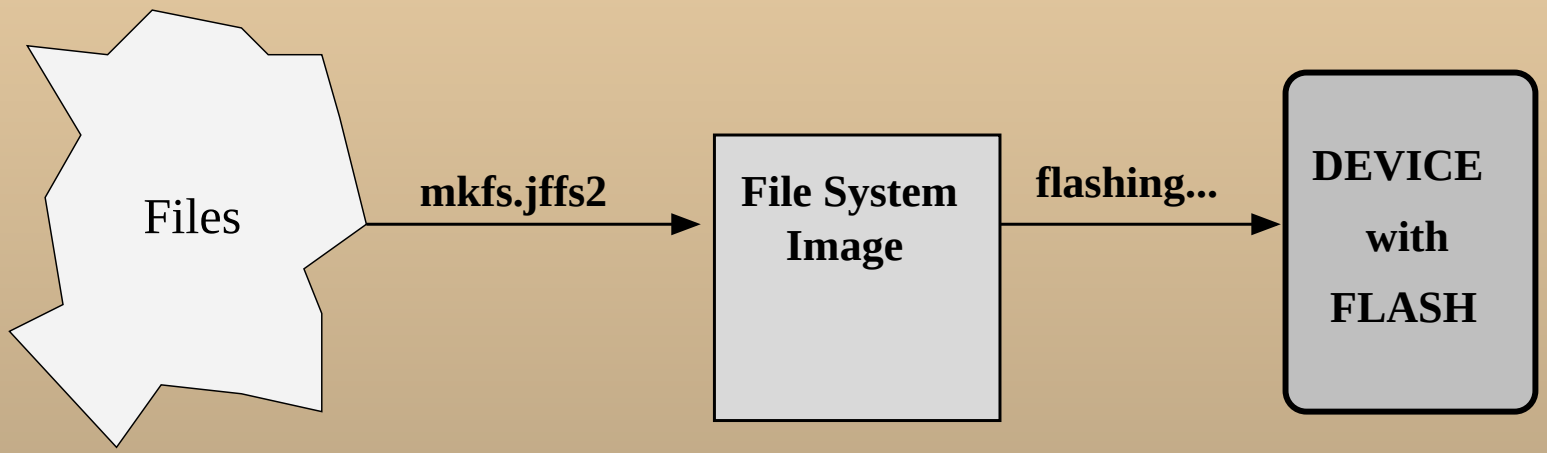
Writes 100 bytes of
'A' into the file (ino 20)
from offset 0

Writes 80 bytes of
'B' into the file (ino 20)
from offset 100

JFFS2

- Journaling Flash File System, version 2
- Splits the information (files) into typ. 4KB blocks
- Compression support - compresses blocks individually
- It uses ZLIB compression library

JFFS2 - usage method



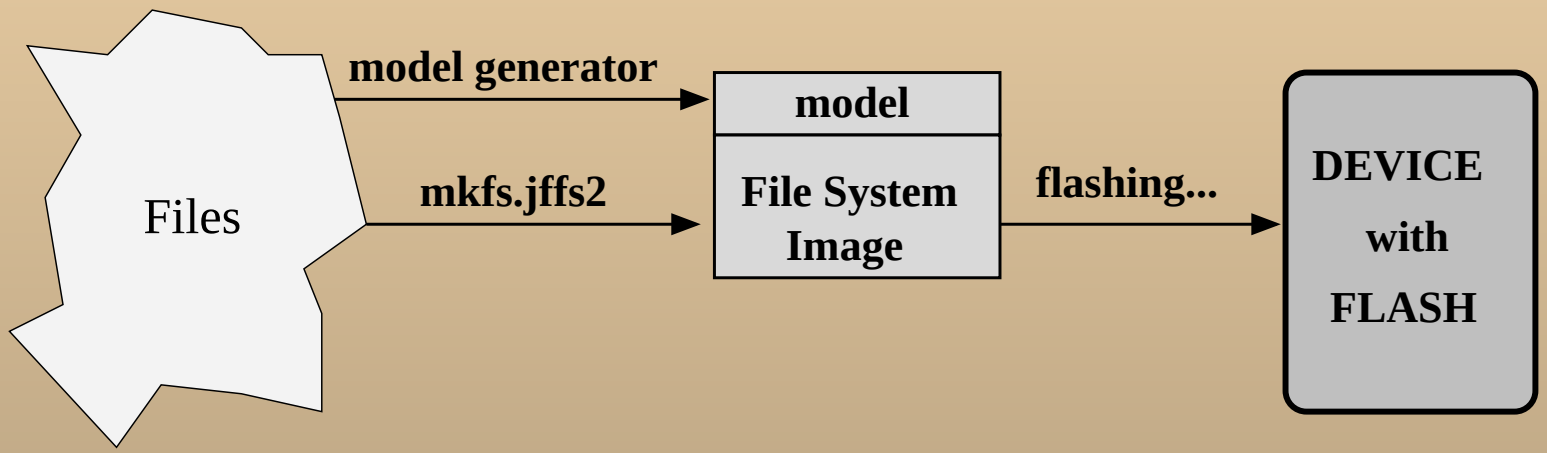
Our improvements

- Replace ZLIB with a compression framwork (compressors are plugins)
- Three compression mode support: none, size, priority (determines which compressor will be used when a block is needed to compress)
- Adopt and insert some free compressors (LZO, LZARI)
- Model file support
- Develop a new model-based compressor for ARM code compression (ARMLIB)

Model file

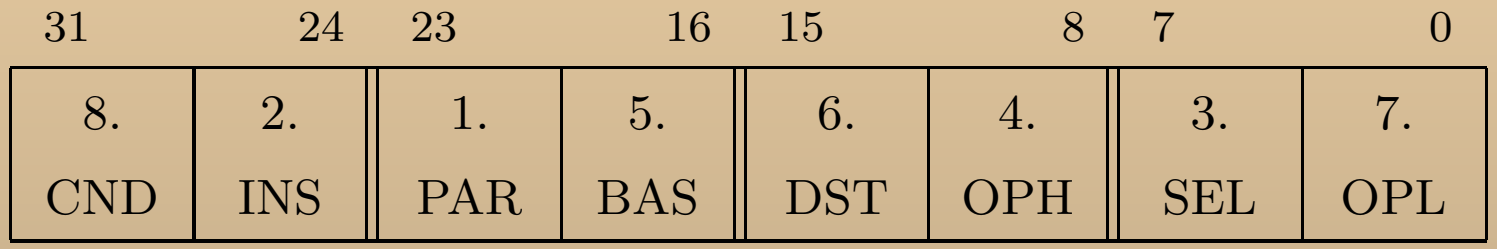
- Non-model-based compressors compress the block individually
- Model-based ones have the possibility to collect some information before the compression of the blocks. This information is available during (de)compressing blocks
- Advantage: better compression ratio can be achieved
- Disadvantage: model(s) must be stored in RAM (but it is much cheaper than flash)

Improved JFFS2 - usage method



ARMLIB

- Designed to compress ARM binary code (32 bit instructions)
- All instruction is split into 8 parts - and reordered. All parts (tokens) are 4 bits length.



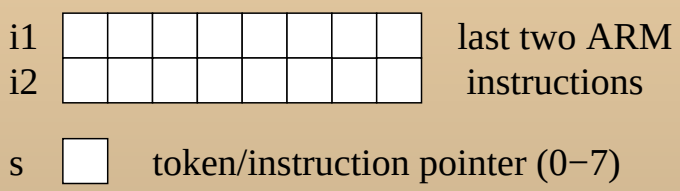
- Tokens are coded by aritmetic coder.
- Model: binary decision tree

The model of ARMLIB

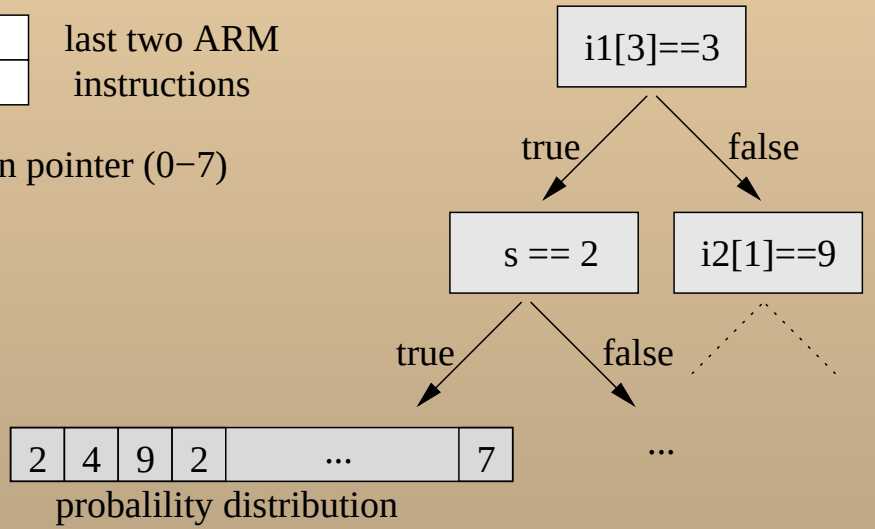
- Leaves are the probability distribution of the tokens - used by the arithmetic coder
- Attributes are the tokens of the previous two instruction, plus one which identifies which part of the instruction is under (de)compressiong
- Decisions nodes: compares an attribute (predictor) with a constant
- The tree is built by an ID3 like algorithm (greedy, entropy based)
- Pruned by a cost based algorithm, where the cost are the sum of the tree storage-size plus the estimated encoded size

Decision tree of ARMLIB

17 Attributes (predictors)



Binary Decision Tree



Results

Compression mode	Image Size	Boot time
no compression	25 145 144	28 sec
ZLIB (original JFFS2)	13 758 396	24 sec
priority (read speed)	14 998 068	22 sec
size	11 153 120	117 sec
size without ARMLIB	13 695 912	34 sec

Familiar Linux 7.2 GPE2 on IPAQ 3970

Open Sourcing

- Discussed on the mailing list of JFFS2/MTD
- Write access to its CVS
- Already committed: compression framework, LZO, LZARI and other small improvements
- Sooner or later it will be taken over into the Linux kernel
- In progress: model file support, ARMLIB
- http://www.inf.u-szeged.hu/jffs2/

Future

- Finish committing all features into CVS
- Optimize ARMLIB
- Develop new compressors (text/XML, ...)
- Other JFFS2 improvements (mount time, ...)