OPEN SOURCE SUMMIT

China 2019

OPEN SOURCE SUMMIT

China 2019

EROFS file system

Gao Xiang <gaoxiang25@huawei.com>

Self-introduction & Background

1. Graduated from university and joined HUAWEI for 3 years, working for OS Kernel Lab Dept., mainly focusing on Linux file system improvement for HUAWEI mobile phones:
   - F2FS support & bugfix;
   - New solutions of requirements from our products:
     1)HUAWEI sdcardfs;
     2)EROFS;
     3)Other useful solutions which are under development...
2. User available space becomes tight with the increase of Android ROM, especially for our low-end devices (some of them are still with 16GB total storage, saving 1~2GB more storage space is useful for users with the help of EROFS);
3. Some solution to leave more space and maintain high-performance to end users?

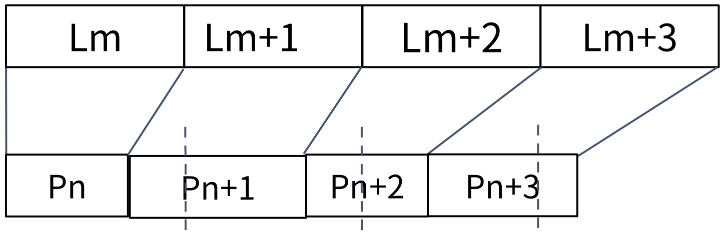
Questions

1. What kind of file types can be compressed practically? and where are they stored in?
   a. Photos, Pictures, Movies? ✗ (no gain to compress losslessly again)
   b. Text material? ? (compressible but the total amount depends on end-user behaviors)
   c. Database? ✗ (some part of overwrite IO patterns could kill the performance)
   d. BINs, shared library, some APKs, preload configuration files? ✓ (mostly read-only files)
2. Is there some solution to resolve our compression requirements?
   squashfs is not good enough for real-time decompression (especially for embedded devices with limited memory, such as mobile phones) since
   1. extra memory overhead;
   2. noticeable read amplification (* based on its default 128KB block size)
   3. some metadata parsing have to be done synchronously limited by its on-disk design.
3. The performance can be even better with compressing effectively in addition to resolving the above squashfs issues, as shown in the following pages.

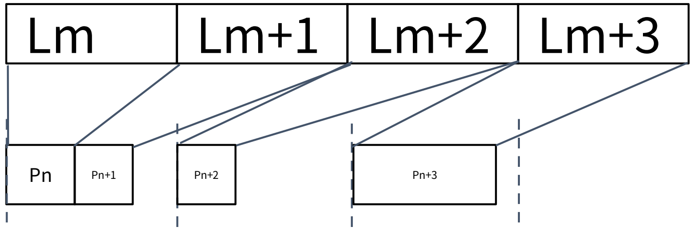
Fixed-output compression

Advantage:

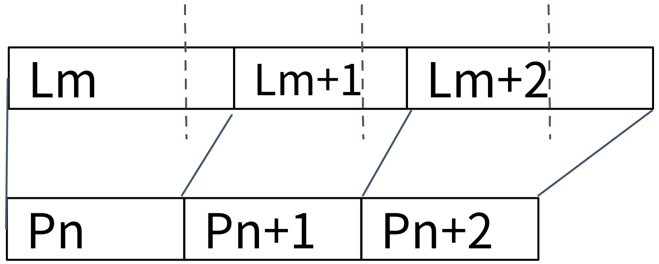
1. improved storage density (high CR -> less IO for most patterns -> read performance improvement);
2. decompression in-place (no memcpy) compared with fixed-input compacted compression;
3. can be on-disk compatible with block boundary aligned compression (eg. Btrfs) in order to archive zero IO read amplification (However, in my opinion, it depends on actual IO patterns and less useful for fixed-output decompression since all compressed data in the block can be compressed.)



fixed-input compacted compression (squashfs, cramfs)



block boundary compression (btrfs)
it is only useful if CR >> 2 (more than 50% off),
it is tough for 4k compress block



fixed-output compression (erofs)

Comparsion of erofs/squashfs/btrfs image sizes

Support 4k compressed cluster only in current erofs kernel implementation due to our requirement:

dataset	testcase	output size	CR
enwik9	enwik9	1,000,000,000B	1
	enwik9_4k.squashfs.img	621,211,648B	1.61
	enwik9_4k.erofs.img	**558,133,248B**	**1.79**
	enwik9_8k.squashfs.img	556,191,744B	1.80
	enwik9_128k.squashfs.img	398,204,928B	2.51
	enwik9_128k.btrfs.img*	657,608,704B	1.52
silesia.tar	silesia.tar	211,957,760B	1
	silesia.tar_4k.squashfs.img	114,524,160B	1.85
	silesia_8k.squashfs.img	106,094,592B	2.00
	silesia.tar_4k.erofs.img	**105,267,200B**	**2.01**
	silesia.tar_128k.squashfs.img	81,768,448B	2.59
	silesia.tar_128k.btrfs.img*	104,628,224B	2.03

compressed with lz4 1.8.3 with command lines:
1) mksquashfs enwik9 enwik9_4k.squashfs.img -comp lz4 -Xhc -b 4096 -noappend
2) mkfs.erofs -zlz4hc enwik9_4k.erofs.img enwik9
* for btrfs, mounting with “nodatasum,compress-force=lzo”, observed with compsize, thus no metadata at all.

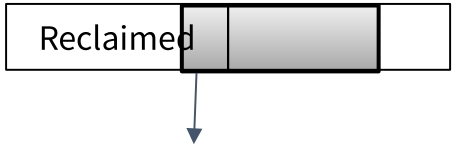
Fixed-output compression (cont.)

Typically, there are 2 ways for compress file system to decompress:

1. read all compressed data into bdev page cache (like all metadata do) and decompress ⇒ squashfs;
2. read compressed data to temporary buffer and decompress ⇒ e.g., btrfs.

However, both 2 ways takes extra overhead since

- For 1), it could cause page cache thrashing, imagine <u>about 50% compressed data of the original file size are added into page cache inactive LRU list at least for embeded devices, many of them are used-once or at a very low frequency;</u> furthermore, it’ s hard to decompress directly if some of compressed pages are reclaimed.
- For 2), it can hardly use the remaining compressed data since temporary buffer could be freed immediately.



compressed block needed to read

Fixed-output compression can make full use of data in compressed block in principle (all compressed data can be decompressed if needed.)

Our solution

EROFS has decompression strategies in 2 dimensions:

(Dimension 1) Cached or In-place IO decompression

- Cached decompression currently for incomplete compression read ;
- In-place IO decompression for complete compression read.

"Complete or incomplete" means whether or not data in the compressed block are all requested.

The policy can be refined later since for small CR it can be fully decompressed in advance since it takes the similar amount of memory if the compressed data are cached in memory.

(Dimension 2) Sync or async decompression

- Sync decompression for small number of page requests, which decompress in its caller thread;
- Async decompression for large read requests, which decompress in workqueue context.

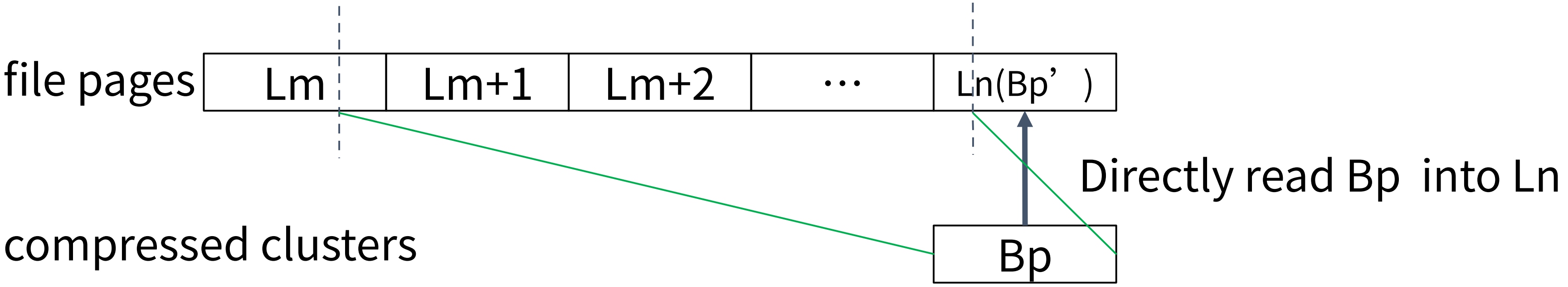
The policy can be refined later as well, since erofs cannot make difference between async/sync readahead in .readpages().

In-place IO decompression

Why we need this? mainly for three reasons:

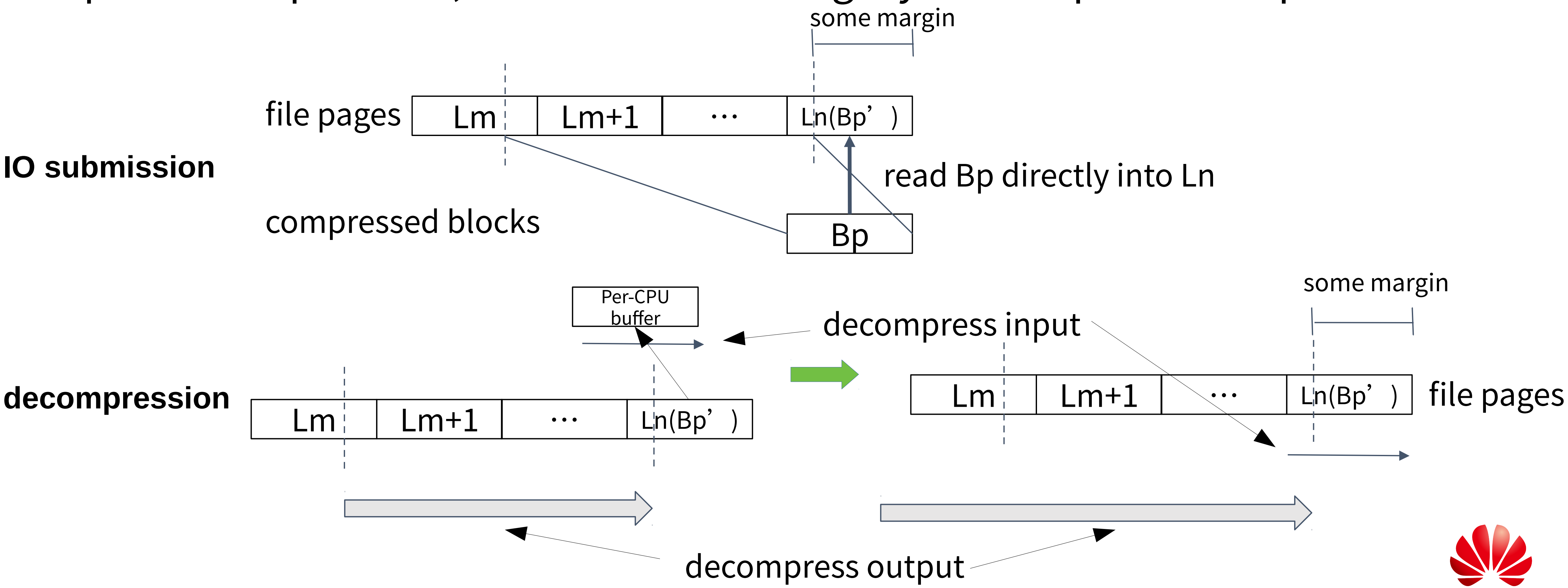
1.doing cached decompression only could cause cache thrashing;
2.considering many read requests are pending for IO ending, temporary buffers cannot be freed. Allocating immediate buffers will cause more memory reclaim as well compared to uncompressed generic file system such as ext4 ⇒ especially for HUAWEI camera heavy memory workload;
3.it will be later used for decompressing in-place.

In this way, compressed data are read in its last decompressed pages as much as possible, as illustrated below:



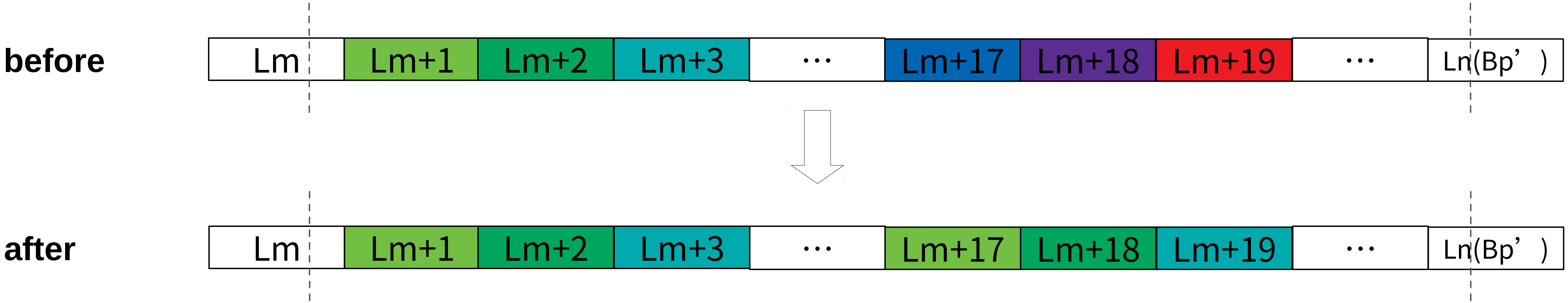
Decompression In-place

As mentioned before, decompression in-place can be implemented with fixed-output decompression, which is hard for legacy fixed-input decompression.



Limited bounced buffers

Since all lz-based compression algorithms use sliding-window technology, EROFS can use limited (64KB for lz4) bounced buffers, which minimize memory consumption as well. Limited bounced buffers is also implemented in new decompression backend.



before

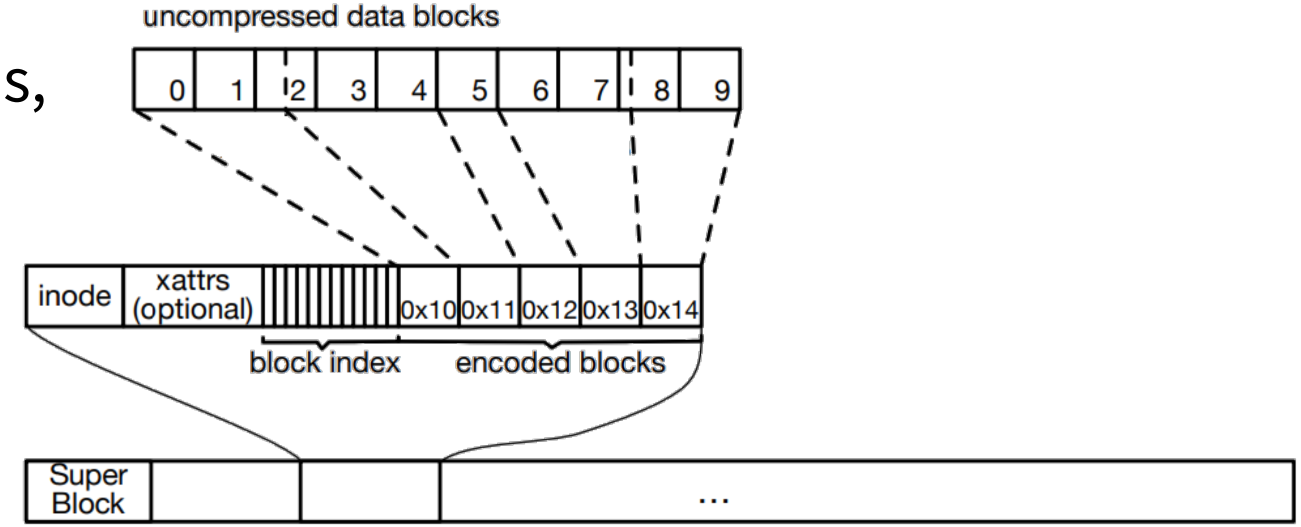
Lm | Lm+1 | Lm+2 | Lm+3 | … | Lm+17 | Lm+18 | Lm+19 | … | Ln(Bp’)

after

Lm | Lm+1 | Lm+2 | Lm+3 | … | Lm+17 | Lm+18 | Lm+19 | … | Ln(Bp’)

On-disk brief introduction

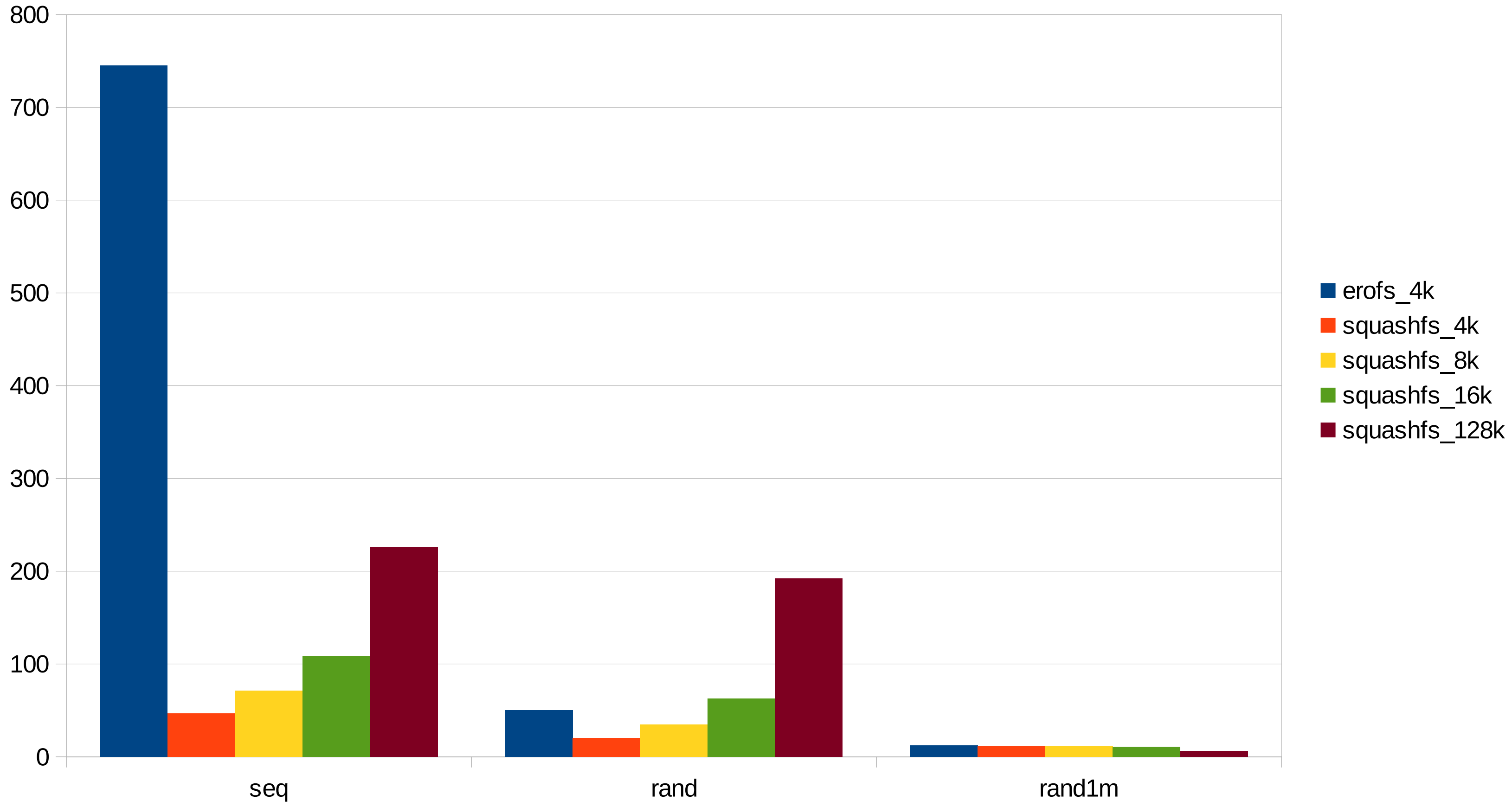
- Little-endian
- 4k block size currently (nobh);
- Mixed metadata with data (In other words, flexible enough for mkfs to play with it);
- 32 (v1) or 64 (v2)-byte inode base size;
- 64 bit + ns timestamp, each file can have its own timestamp for v2;
- Support tail-end data inline;
- Support XATTR, POSIXACL (5.2+);
- Support statx (5.3+);
- Support compacted indexes (5.3+);
- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/staging/erofs/Documentation/filesystems/erofs.txt?h=v5.1



Microbenchmark

CPU: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz (4 cores, 8 threads) DDR: 8G
SSD: INTEL SSDPEKKF360G7H

enwik9 FIO (psync, 1 thread)



Real app launching

CPU: MT6765 (8 Cortex-A53 cores) DDR: 2GB eMMC: 32GB

App. #	1	2	3	4	5	6	7	8	9	10	11	12	13
w/o FIO workload	-16.4	-3.5	+4.2	-4.0	-7.5	-1.4	-6.8	+6.3	-2.2	-18.4	-3.3	-7.6	-4.5
w/ FIO workload	-2.8	-12.9	-5.4	+3.9	-7.6	+3.7	+4.4	-2.6	+9.9	+4.0	-11.1	-10.3	-15.1

Relative boot time of thirteen applications. Each number in the table is the average value of at least five boots. Negative numbers show the boot time reduction (in %) compared with Ext4, while positive numbers indicate the boot time is prolonged (in %).
FIO workloads as the background workload: four threads randomly read and write individual files with rate limited at 256KB/s for both reads and writes.

EROFS upstream progress

- Accepted as a staging driver from Linux 4.19 (2018/07);
- Decompression in-place is pending to be merged for Linux 5.3 (2019/06);
- Trying to move out of staging from Linux 5.4...

EROFS landing progress

1. LZ4 ARM64 assembly optimization is already used for our products as well;
2. All of new products starting from EMUI 9.0.1 (Android Q) are used EROFS file system;
3. Several million phones shipped with erofs on the market, including P30 & P30 pro (EMUI 9.1);
4. Decompression inplace will be used in our products of course;
5. Hoping to upstream to Android Open Source Project later as well, but… It needs time…
6. Hoping it will be used in wider scenarios...

My to-do lists on EROFS

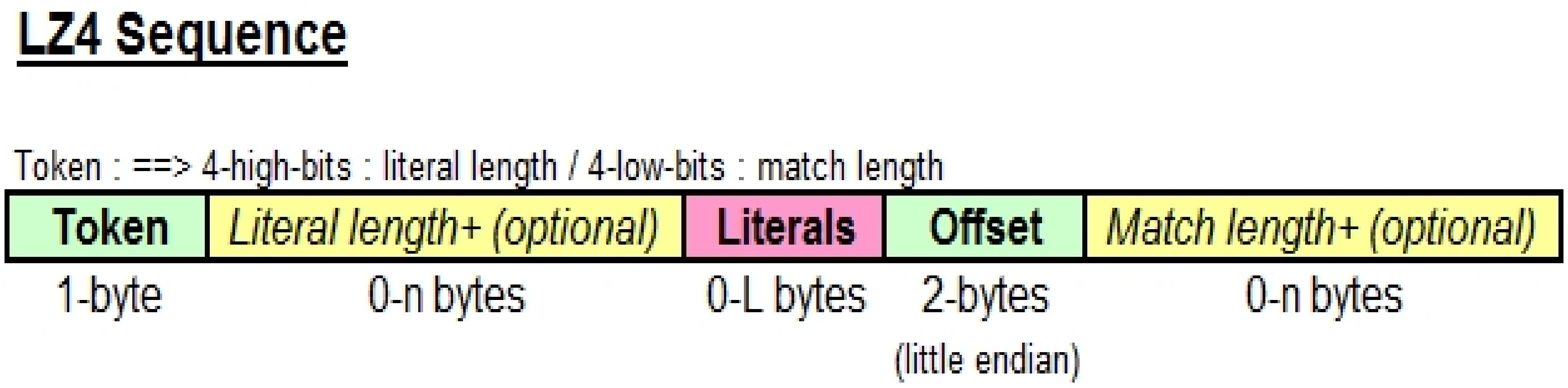
1. Hoping to move into fs/ directory as a replacement of squashfs for performance scenarios;
2. Tuning the decompress performance even further;
3. Supporting larger cluster size for output-fixed decompression for guys who really need it;
4. erofs-FUSE to support old Linux kernel and other platforms.

References

1. linux-erofs mailing list: linux-erofs@lists.ozlabs.org
2. erofs-utils userspace tools: git://git.kernel.org/pub/scm/linux/kernel/git/xiang/erofs-utils.git
3. [PATCH v3] decompression inplace approach: https://lore.kernel.org/lkml/20190624072258.28362-1-hsiangkao@aol.com/

Thank you

LZ4 format and safe margin



LZ4 Sequence

Token : ==> 4-high-bits : literal length / 4-low-bits : match length

Token	*Literal length+ (optional)*	Literals	Offset	*Match length+ (optional)*
1-byte	0-n bytes	0-L bytes	2-bytes (little endian)	0-n bytes

Decompression inplace lz4 upstream commit and discussion between Yann Collet and me:
https://github.com/lz4/lz4/commit/b17f578a919b7e6b078cede2d52be29dd48c8e8c
https://github.com/lz4/lz4/commit/5997e139f53169fa3a1c1b4418d2452a90b01602

Picture taken from https://www.programering.com/a/MzN5QjNwATg.html