Whamcloud

Efficient Storage Utilization Using Client-side Data Compression

Artem Blagodarenko

Let’s picturize

Client-side data compression

Whamcloud

- **Client-side data compression** (CSDC) is a new feature for the Lustre file system. This feature provides **transparent** (for users, configured by admins) compression and decompression of data stored on the file system.
- The feature **is simple to use** (once activated) and can be set on a per-file/directory or per-component basis (**each component** has its own compression parameters).
- Multiple compression algorithms are supported (**lzo**, **lz4**, **lz4hc** and **gzip** now). **Easy to expand** framework to add new compression algorithms.
- Compression **scales with multiple clients**. Each request is being compressed independently, so the process is already parallelized and CPUs cores scalable.
- Does **not** depend on ZFS compression (like in Hamburg University's project). No changes to LDISKFS on-disk data structures. Compression *mostly* (see next slides) done on the client side.

Usage

- Early adopters can enable CSDC with the following command:

  ```
  sudo lustre/utils/lctl set_param -n llite.*.enable_compression 1
  ```

  - This can be made permanent by utilizing the -P option for the set_param command.
- The lfs setstripe command is used to configure compression on the file system:

```
lfs setstripe –E eof -Z lz4:5 --compress-chunk=512k <dir|new_file>
$ lfs getstripe <file> | grep compr
lcme_compr_type: lz4
lcme_compr_lvl: 5
lcme_compr_chunk_kb: 512
lmm_pattern: raid0,compress
```

- -Z or --compress

  Accepts 2 values separated by a colon

  First value: compression algorithm
  - lzo (no compression level accepted)
  - lz4
  - lz4hc
  - gzip

  Second value: compression level (optional)
  - Integer values: 0 through 9
  - (0) fastest compression time
  - (9) best compression ratio

  Example:
  - -Z lz4:5
- --compress-chunk
  - Defines compression chunk size
  - Accepts string values
  - Minimum value: 64k
  - Maximum value: maximum stripe size
  - Default value: 64k

Compression aware tools

lfs find - used to search the lustre file system like 'POSIX' find with additional options (see lfs find man page for those options)

- `--comp-flags=[^]compress` to locate file with/without compressed components
- `--comp-flags=[^]nocompr` to locate file with/without setting component compress preference
- `[!] --layout=compress` to locate file with/without compressed components
- `[!] --compress-type=<type>` find files with/without specified compress algorithm
- `[!] --compress-level=[+-]<level>` find files with/without specified compress level

Examples:

Find already compressed files

```
$ lfs find --comp-flags=compress <dir>
```

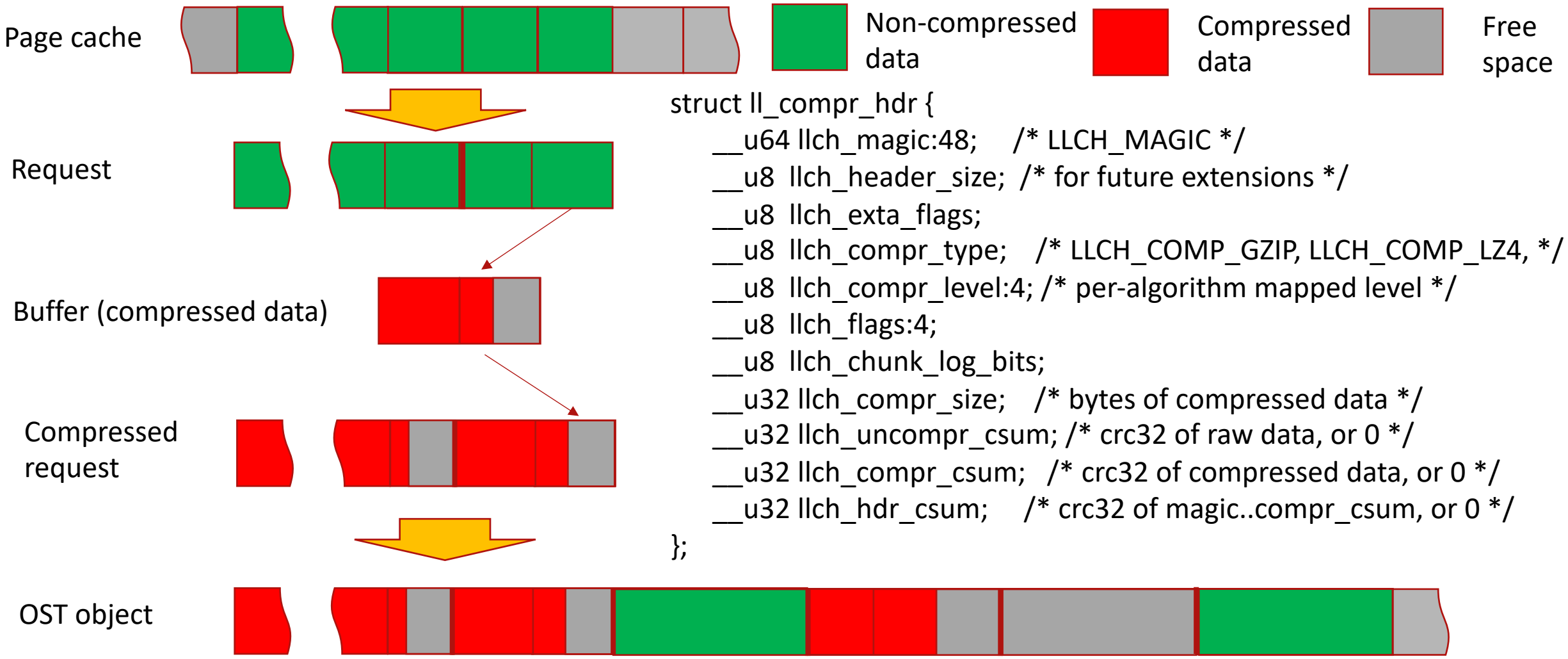
Find compressed files with type other than `lz4` (e.g. to recompress with `lz4` in background)

```
$ lfs find --compress-type=^lz4 <dir>
```

Find compressed files with level < 5 (e.g. to recompress to a higher level)

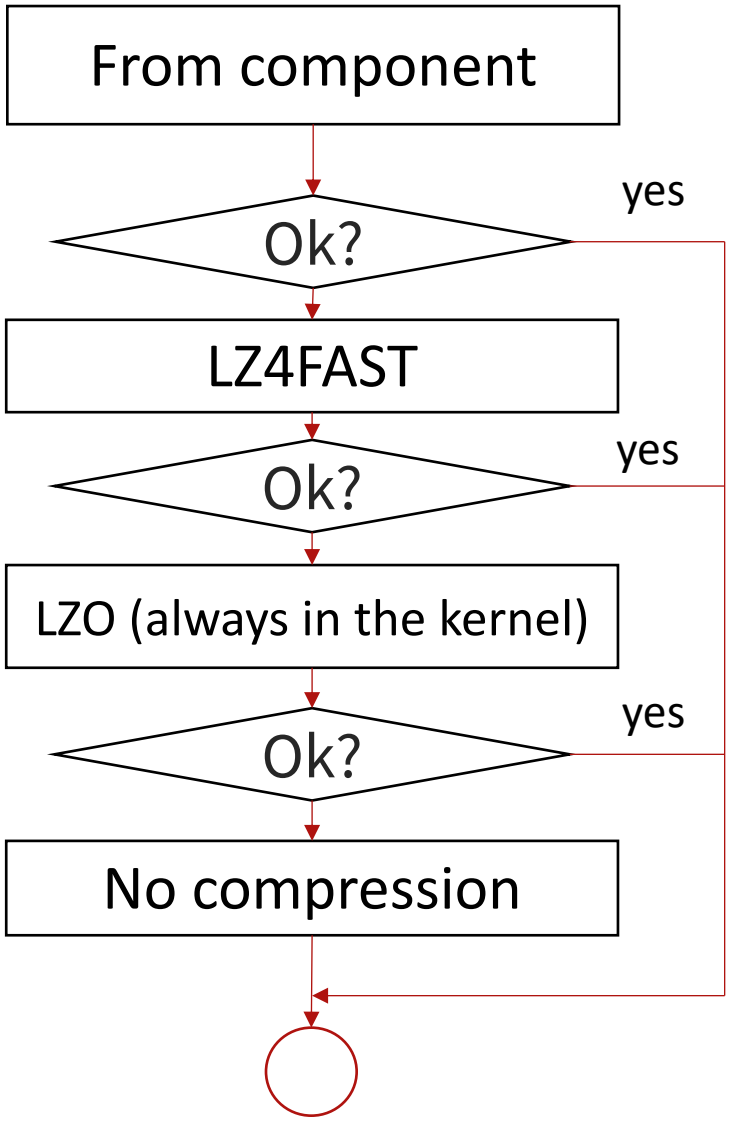
```
$ lfs find --compress-level=-5 <dir>
```

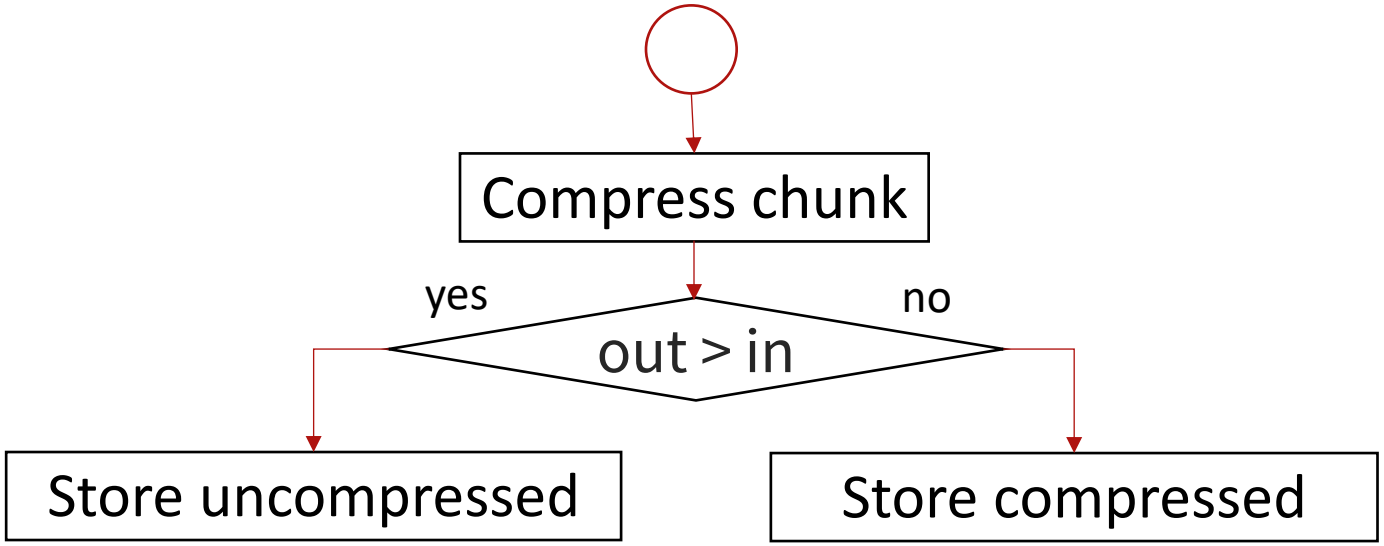
Data Compression Scheme




```
struct ll_compr_hdr {
        __u64 llch_magic:48;     /* LLCH_MAGIC */
        __u8  llch_header_size;  /* for future extensions */
        __u8  llch_exta_flags;
        __u8  llch_compr_type;    /* LLCH_COMP_GZIP, LLCH_COMP_LZ4, */
        __u8  llch_compr_level:4; /* per-algorithm mapped level */
        __u8  llch_flags:4;
        __u8  llch_chunk_log_bits;
        __u32 llch_compr_size;    /* bytes of compressed data */
        __u32 llch_uncompr_csum; /* crc32 of raw data, or 0 */
        __u32 llch_compr_csum;   /* crc32 of compressed data, or 0 */
        __u32 llch_hdr_csum;     /* crc32 of magic..compr_csum, or 0 */
};
```

Compression algorithms





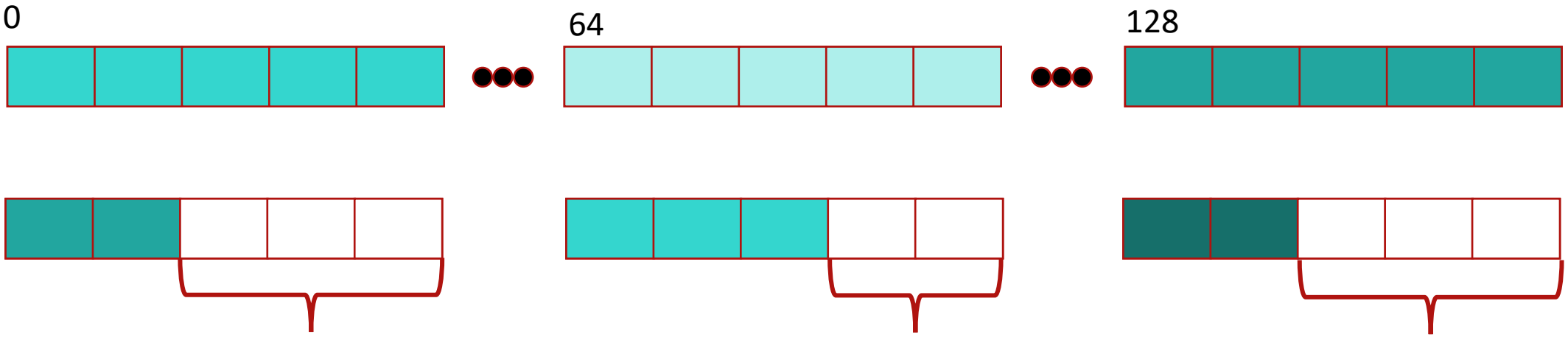
- In-kernel LZO algorithm **always available** for all client kernel versions
- LZ4/LZ4H, GZIP deployed with kernel or Lustre FS (preferable)
- Up to 255 compression types are available for extention
- Any heuristic for compression params is possible

LDISKFS allocator changes for improved data density

Whamcloud

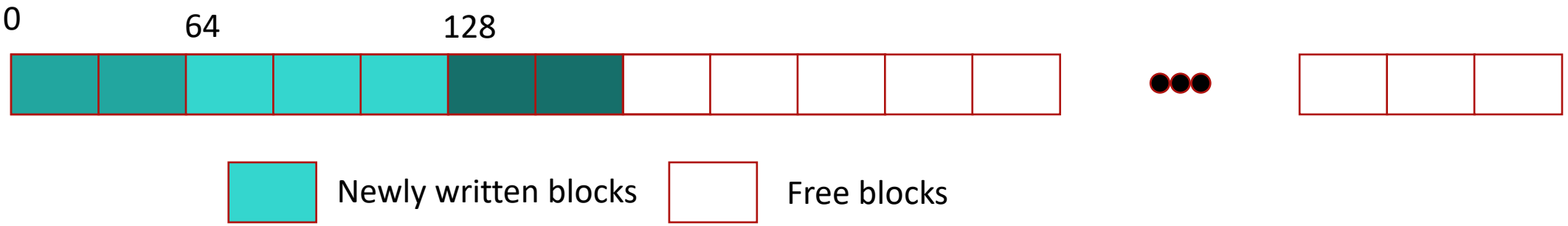
- Compression will always reduce data size by at least one 4KB block, or it is skipped
- OST will write chunks starting at file logical offset for each chunk to LDISKFS
  - Client must read and write **whole chunks** starting at an even multiple of the chunk offset
- From LDISKFS perspective compressed chunks have holes between them in file/block allocation
  - For example, 64KiB chunk compressed to 24KiB the next chunk will have a 40KiB "hole" from LDISKFS logical offset perspective
- Optimize on-disk blocks to be contiguous
- Client sends `OBD_BRW_COMPRESSED` flag with each compressed write RPC
- Flag informs allocator that holes will never be filled, and should pack chunks densely

LDISKFS changes to avoid allocation holes in files



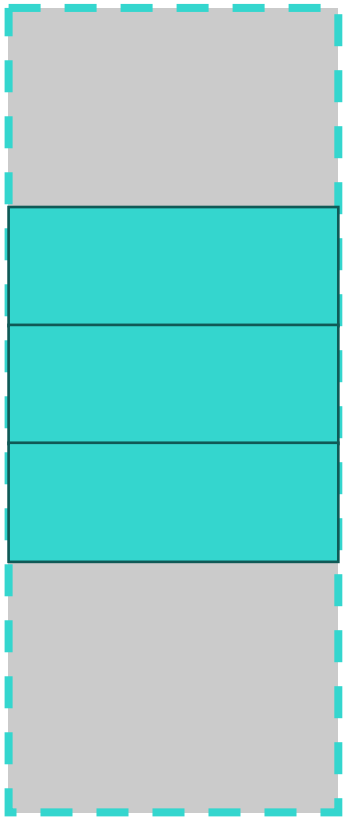
LDISKFS optimization. These blocks normally reserved for multi-client interleaved writes, but in case of compression these blocks will be unused. Gaps decrease read performance (for **HDDs**) and add fragmentation.

```
LDISKFS receives OBD_BRW_COMPRESSED flag and disables the optimization.
Blocks are being written sequentially. This optimizes writing and reading.
```



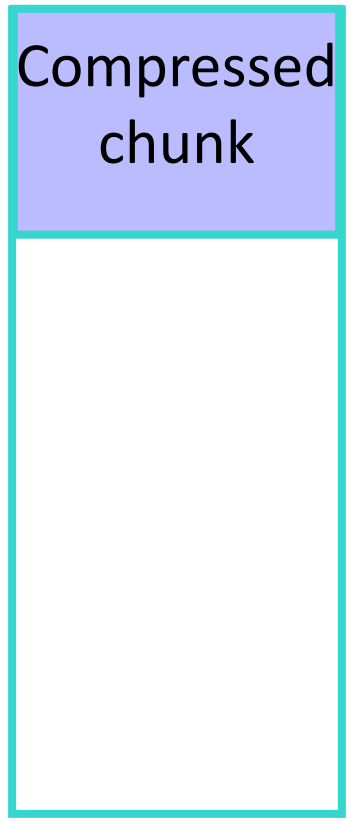
Partial Chunk Operations Problem

chunk

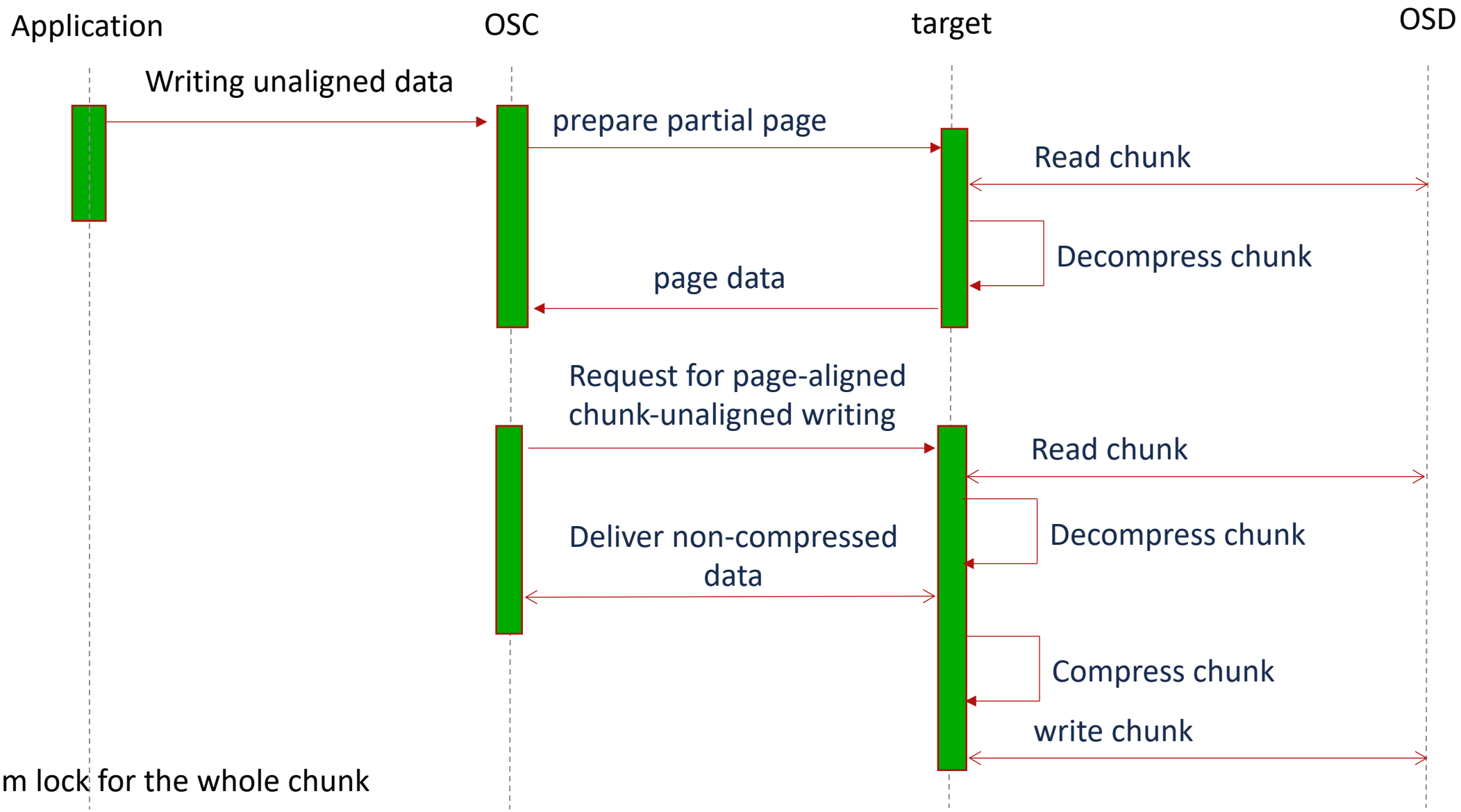


- Read/write is always chunk-aligned
- Reading request extended to the chunk bounds using readahead
- Write path is complicated: once data written, the whole chunk should be changed on a rewrite
- All the data for the compression chunk must be in the page cache to do compression
- All required pages should be read, and they will be clean, non-dirty cache pages can be cleared from cache at any time
- In case witting data is not page-aligned, ll_prepare_partial_page() reads the rest of the page from disk

disk

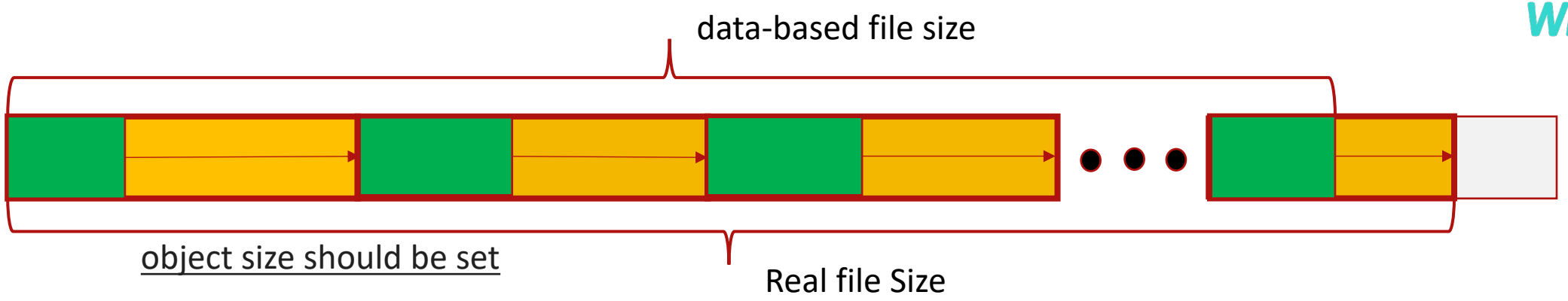


Partial Chunk Rewrite Server-Side Solution



*ldlm lock for the whole chunk

Some CSDC-related details

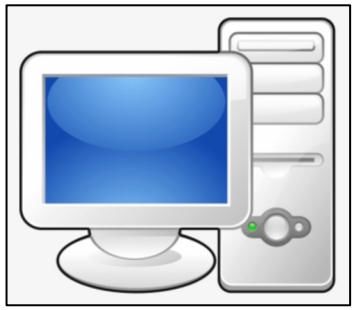


- Store **compression info** in LMA2 EA(on the **server**). Useful for utilities like a LFSCK.
- Allow **raw acces**s to compressed data for *lfs migrate.*
- LU-16837 handle unknown layout component. If lustre **client** encounters **unknown layout component** pattern in a mirror file, it makes client mark this mirror as invalid and skip it.
- Disable CPU-access features for **RDMA** only pages.
- Add support for **encryption plus compression** – compression (obviously) goes first!
- Avoid picking compressed mirror for **older client** which does not support compressed layout component.

Testing. Configuration

Whamcloud

```
# lscpu
Architecture:          x86_64
  CPU op-mode(s):      32-bit, 64-bit
  Address sizes:       52 bits physical, 57 bits virtual
  Byte Order:          Little Endian
CPU(s):                224
  On-line CPU(s) list: 0-223
Vendor ID:             GenuineIntel
  Model name:          Intel(R) Xeon(R) Platinum 8480CL
    CPU family:        6
    Model:             143
    Thread(s) per core: 2
    Core(s) per socket: 56
    Socket(s):         2
    Stepping:          7
    CPU max MHz:       3800.0000
    CPU min MHz:       800.0000
    BogoMIPS:          4000.00
```



```
~]# cat /etc/es_install_version
EXAScaler SFA Rocky AI400X2
[root@ai400-004 ~]# lctl get_param version
version=2.14.0_ddn101_8_g02a1f63

llite.*.max_read_ahead_mb" = 2048
"obdfilter.*.brw_size" = 16
"osc.*.max_dirty_mb" = 512
"osc.*.max_pages_per_rpc" = "1M"
"osc.*.max_rpcs_in_flight" = 8
"osd-ldiskfs.*.read_cache_enable" = 0
"osd-ldiskfs.*.writethrough_cache_enable" =
0
```

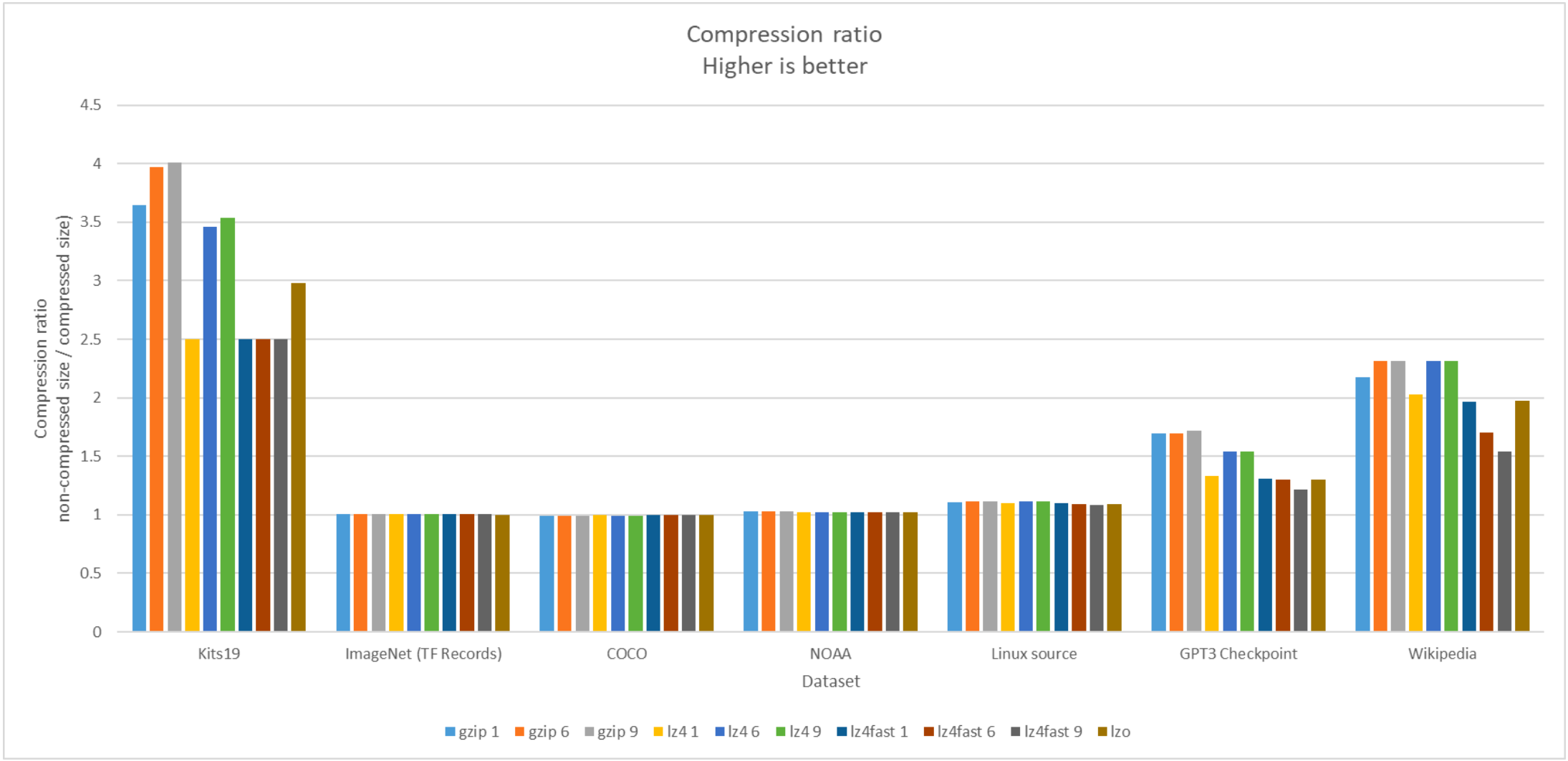
Testing. Datasets

Whamcloud

Name	Link	Area
Kits19	https://github.com/neheller/kits19	Image segmentation (medical)
ImageNet (TF Records)	https://image-net.org/challenges/LSVRC/2012/2012-downloads	Image classification
COCO	https://cocodataset.org/#download	Object detection (heavy weight)
NOAA	https://nomads.ncep.noaa.gov/pub/data/nccf/com/gfs/prod/gdas.20230912/	Weather data
Linux source	https://github.com/torvalds/linux	Source code
GPT3 Checkpoint	https://huggingface.co/TurkuNLP/gpt3-finnish-13B	AI/ML
Wikipedia	https://drive.google.com/drive/u/0/folders/1oQF4diVHNPCclykwdvQJw8n_VIWwV0PT	NLP

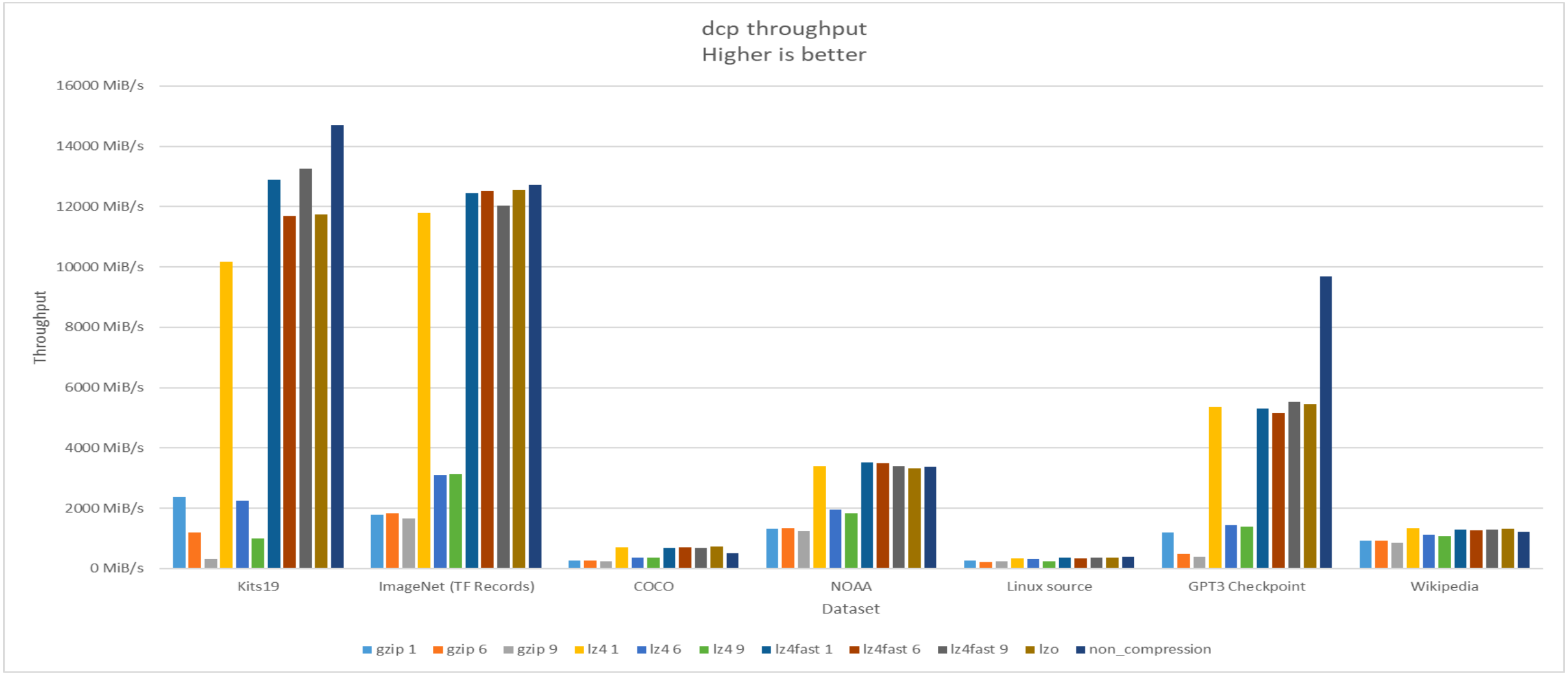
Compression ration

Whamcloud



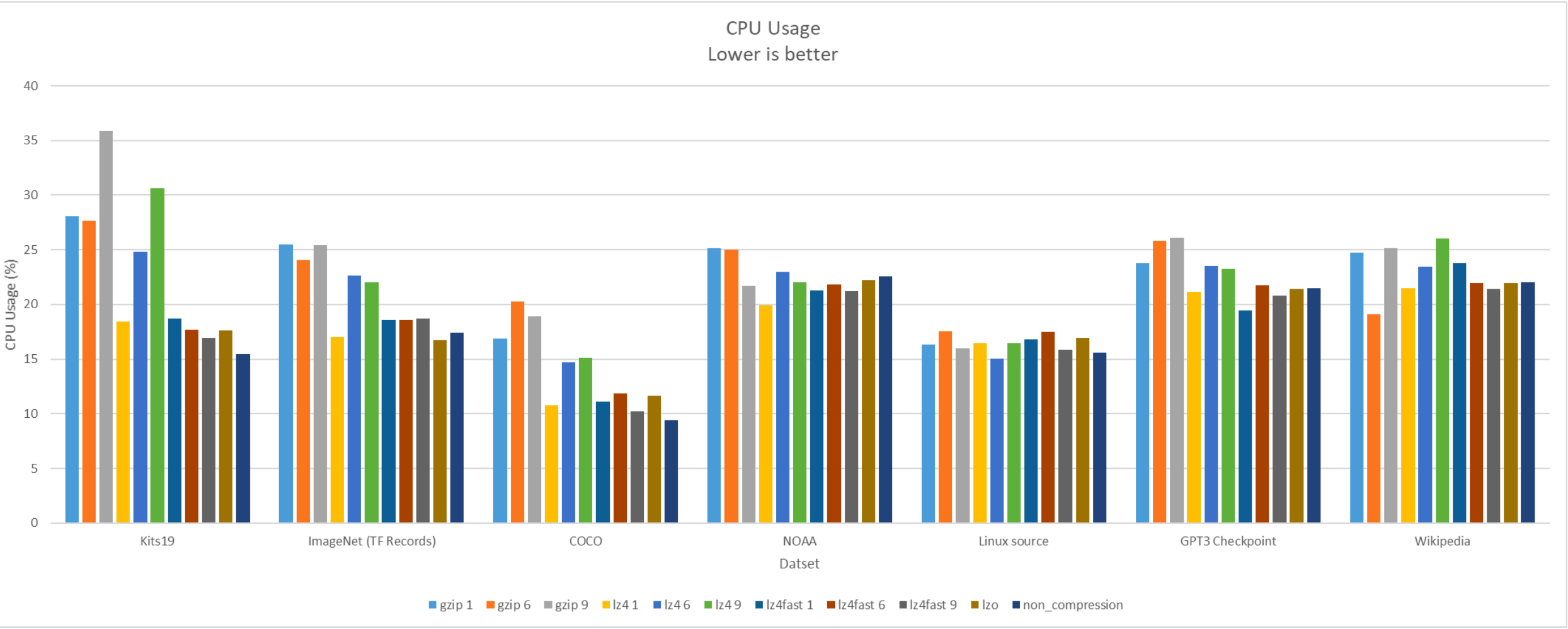
Throughput

Whamcloud



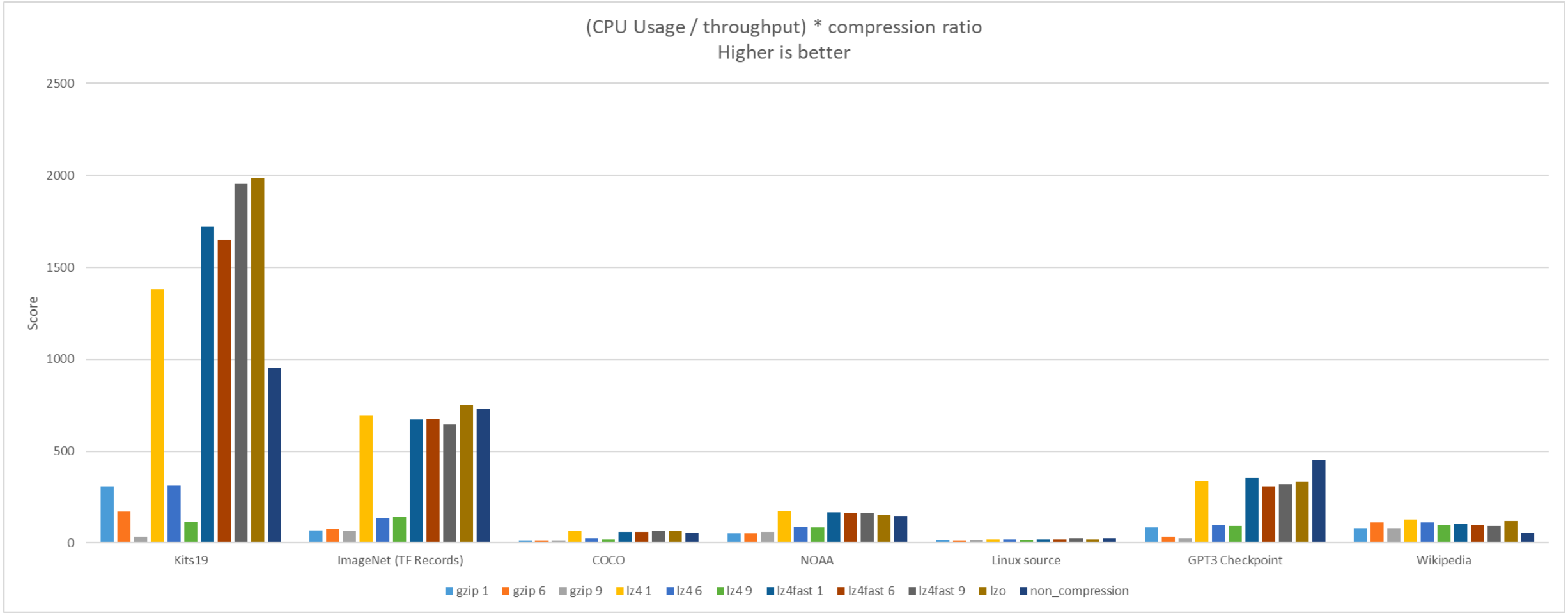
Throughput/CPU

Whamcloud



(CPU Usage / throughput) * compression ration

Whamcloud



Testing results observations

Whamcloud

- The approach **works**.
- Doing buffered, single threaded I/O, compression will have **limited performance costing**.
- There are data types that **leverage** from CSDC **more than others**.
- Compression doesn't improve throughput now. This **will be** optimized though (LU-16897)
- There is known issue that requires to skip the last chunk compression. It is fixed now but hasn’t present in the testing.
- Larger chunks improve compression, but higher read-modify-write overhead

Status

Whamcloud

- The feature is planned to be available with the Lustre FS 2.17 release
- Major functionality development is finished
- Prior testing is finished, more enhancing testing it ongoing

Thank You!
Questions?