Whamcloud

Lustre Client-Side Data Compression

Xu Zhenyu

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* 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>
```

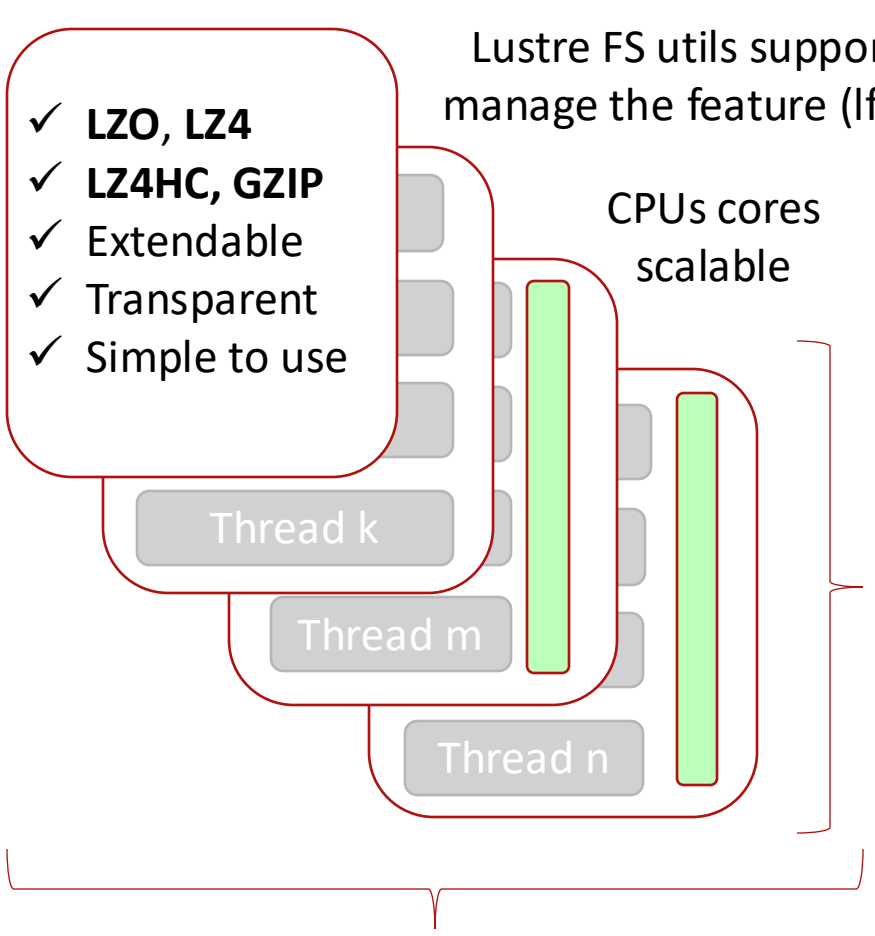
Find compressed files with type other than `lz4` (e.g. to recompress with `lz4` in background)

```
$ 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>
```

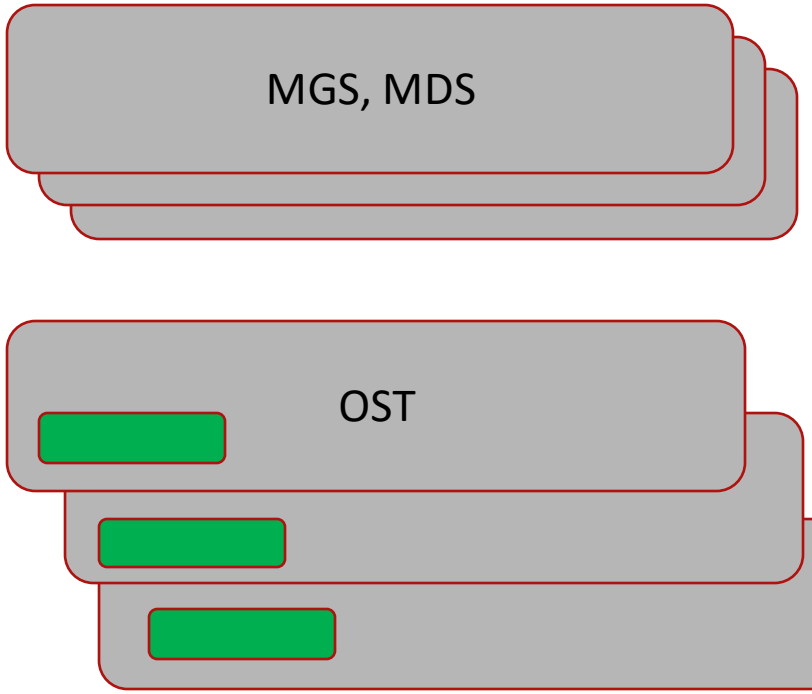
Client-Side Data Compression



Lustre FS utils support to manage the feature (lfs etc)

CPUs cores scalable

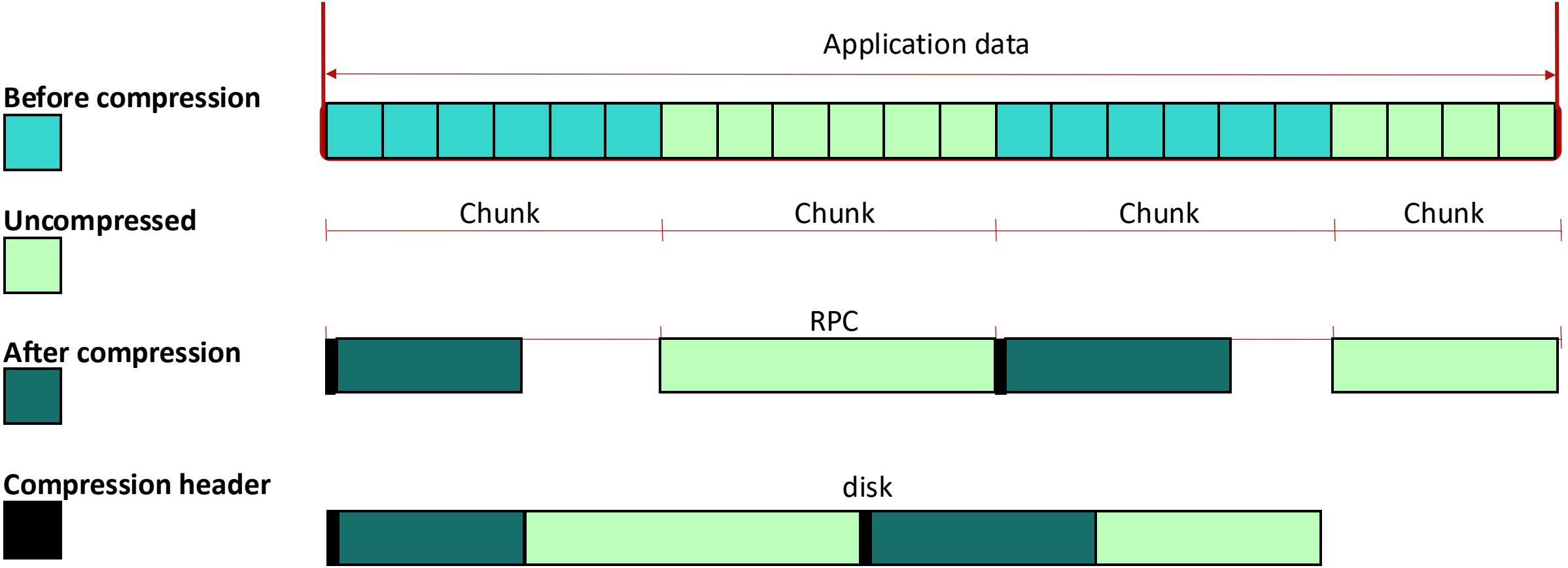
Scales with multiple clients



All advantages of Lustre FS scalable data storage

Data is uncompressed in the client's cache and stored compressed on the OSTs

Data Compression Scheme

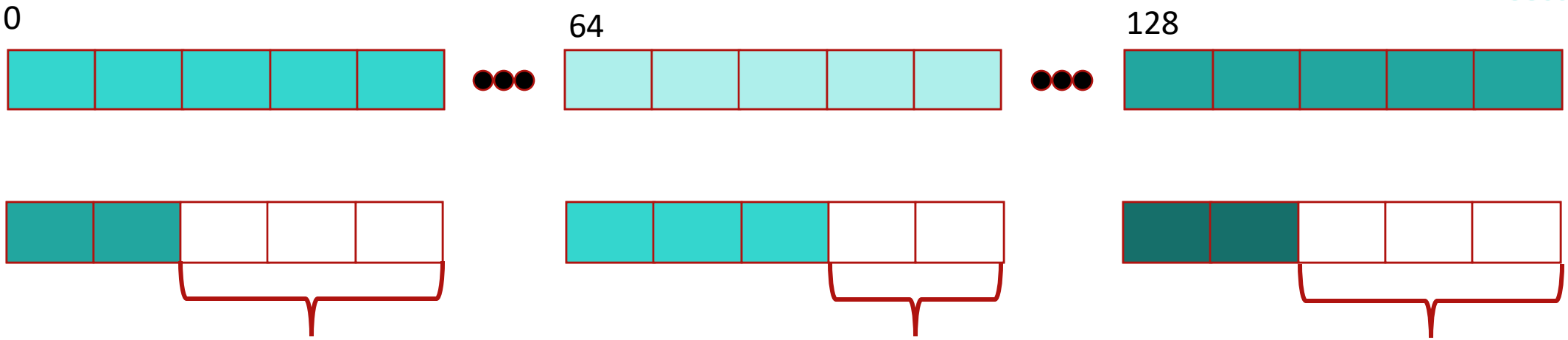


LDISKFS allocator changes for improved data density

Whamcloud

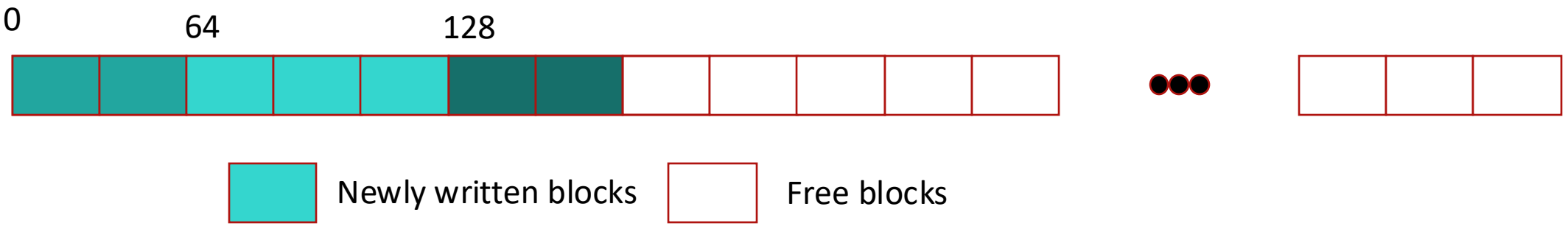
- Compression will always reduce data size by at least one 4KB block, or it is skipped
- OST will write chunks starting at file logical offset for each chunk to LDISKFS
  - Client must read and write **whole chunks** starting at an even multiple of the chunk offset
- From LDISKFS perspective compressed 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



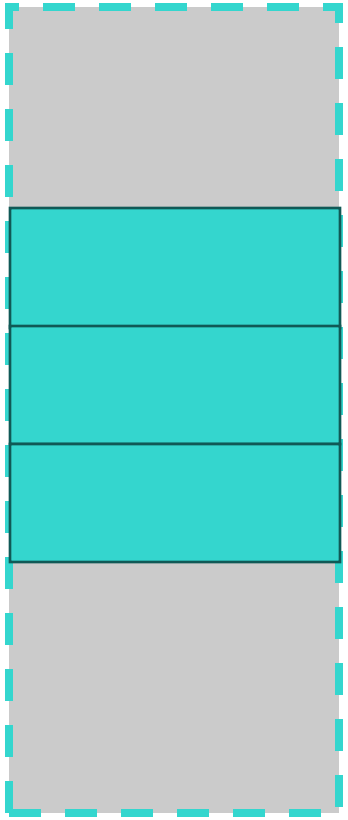
LDISKFS optimization. These blocks normally reserved for multi-client interleaved writes, but in case of compression these blocks will be unused. Gaps decrease read performance (for **HDDs**) and add fragmentation.

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



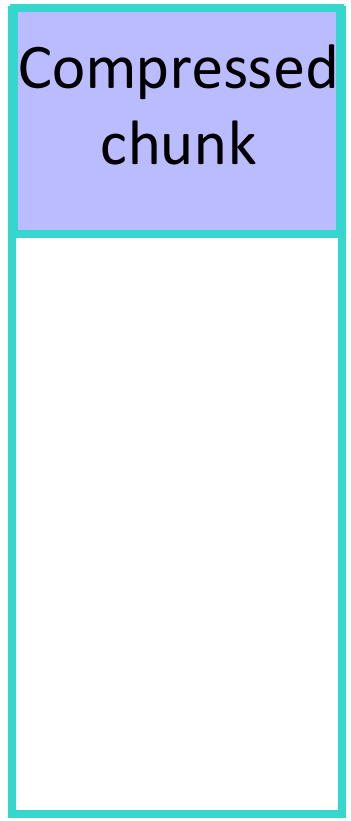
Partial Chunk Operations Problem

chunk

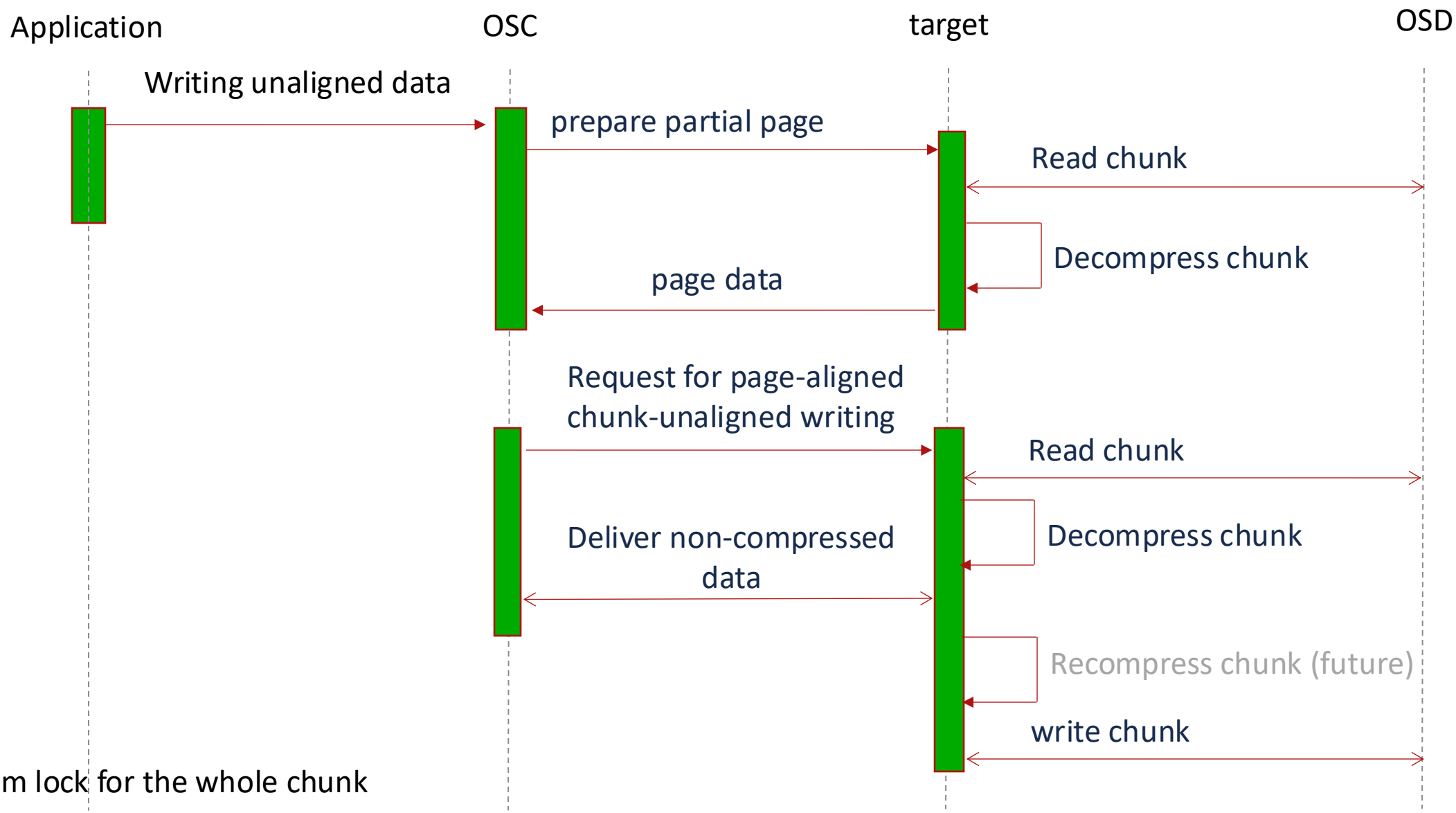


- Read/write is always chunk-aligned
- Reading request extended to the chunk bounds using readahead
- Write path is complicated: once data written, the whole chunk should be changed on a rewrite
- All the data for the compression chunk must be in the page cache to do compression
- All required pages 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

Persistent server-side counters

Whamcloud

- OFD saves compression stats every 10 minutes and at unmount
- Load the saved stats at mount time
- Presents overall compression usage
- The file is called "compr_stats“
- Stored in the root directory of underlying fs and uses the YAML format

```
- { name: uncompressed_objects, count:0, min:92, max:0, sum:0 }
- { name: compressed_objects, count:1, min:92233, max:0, sum:0 }
- { name: compr_chunks_read_disk, count:0, min:9223, max:0, sum:0 }
- { name: compr_bytes_read_disk, count:0, min:92233, max:0, sum:0 }
- { name: compr_bytes_read_user, count:0, min:92233, max:0, sum:0 }
- { name: compr_chunks_read_decompressed, count:0, min:9, max:0, sum:0 }
- { name: compr_bytes_read_decompressed, count:0, min:9, max:0, sum:0 }
- { name: compr_chunks_write_disk, count:4, min:1, max:8, sum:11 }
- { name: compr_bytes_write_disk, count:4, min:8192, max:1310, sum:20}
- { name: compr_bytes_write_user, count:4, min:8192, max:10486, sum:1}
```

Client-side statistics


```
# lfs setstripe -i 0 -c 1 -E eof -Z lz4:6 --compress-chunk=128k /mnt/lustre/testfile

# dd if=/dev/zero of=/mnt/lustre/testfile bs=1M count=4 conv=fdatasync
4+0 records in
4+0 records out
4194304 bytes (4.2 MB, 4.0 MiB) copied, 0.0310476 s, 135 MB/s

# du -h /mnt/lustre/testfile
132K /mnt/lustre/testfile

# lctl get_param osc.lustre-OST0000*.stats_compr
osc.lustre-OST0000-osc-ffff565948a4a000.stats_compr=
write_pages_compr: 1024
write_pages_uncompr: 0
write_chunks_compr: 32
write_chunks_no_compr: 0
write_bytes_compr: 17792
write_bytes_raw: 4194304
write_bytes_incompr: 0
read_pages_compr: 0
read_chunks_compr: 0
read_bytes_compr: 0
read_bytes_raw: 0
```

Detecting and avoiding incompressible data

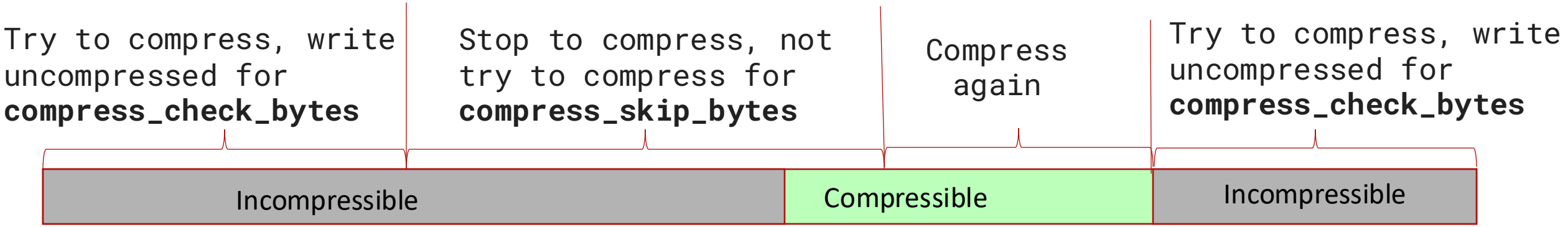
Whamcloud

The **reduced_ratio** (original_size/compress_reduced_size) represents the minimum fraction of pages that are compressed out of each chunk

Compressed chunk needs to shrink by at least 1/reduced_ratio blocks for it to be "compressible".

After every **compress_check_bytes** of compressed data written, the file's compressibility would be re-calculated.

If not compressing, recheck after **compress_skip_bytes * factor**
If data still incompressible, double **factor** before next check.



Incompressible data heuristics in operation

Whamcloud

```
# lctl set_param osc.lustre*.compress_check_bytes=1M
# lctl set_param osc.lustre*.compress_skip_bytes=8M
# lctl set_param osc.lustre*.compress_reduced_ratio=16
# lfs setstripe -E-1 -c1 -Z lz4:4 --compress-chunk=64 /mnt/lustre/foofile
# lctl set_param osc.*.stats_compr=0
# cp -a lustre/tests/AMSR_E_L3_DailyOcean_V05_20111003.hdf.bz2 /mnt/lustre/foofile
# lctl get_param osc.lustre*.stats_compr

osc.lustre-OST0000-osc-ffff565948a4a000.stats_compr=
write_pages_compr: 0
write_pages_uncompr: 861
write_chunks_compr: 0
write_chunks_no_compr: 54
write_bytes_compr: 0
write_bytes_raw: 3526254
write_bytes_incompr: 3526254
read_pages_compr: 0
read_chunks_compr: 0
read_bytes_compr: 0
read_bytes_raw: 0
```

```
#du -h /mnt/lustre/foofile
3.4M /mnt/lustre/foofile

#du -h lustre/tests/AMSR_E_L3_DailyOcean_V05_20111003.hdf.bz2
3.4M lustre/tests/AMSR_E_L3_DailyOcean_V05_20111003.hdf.bz2
```

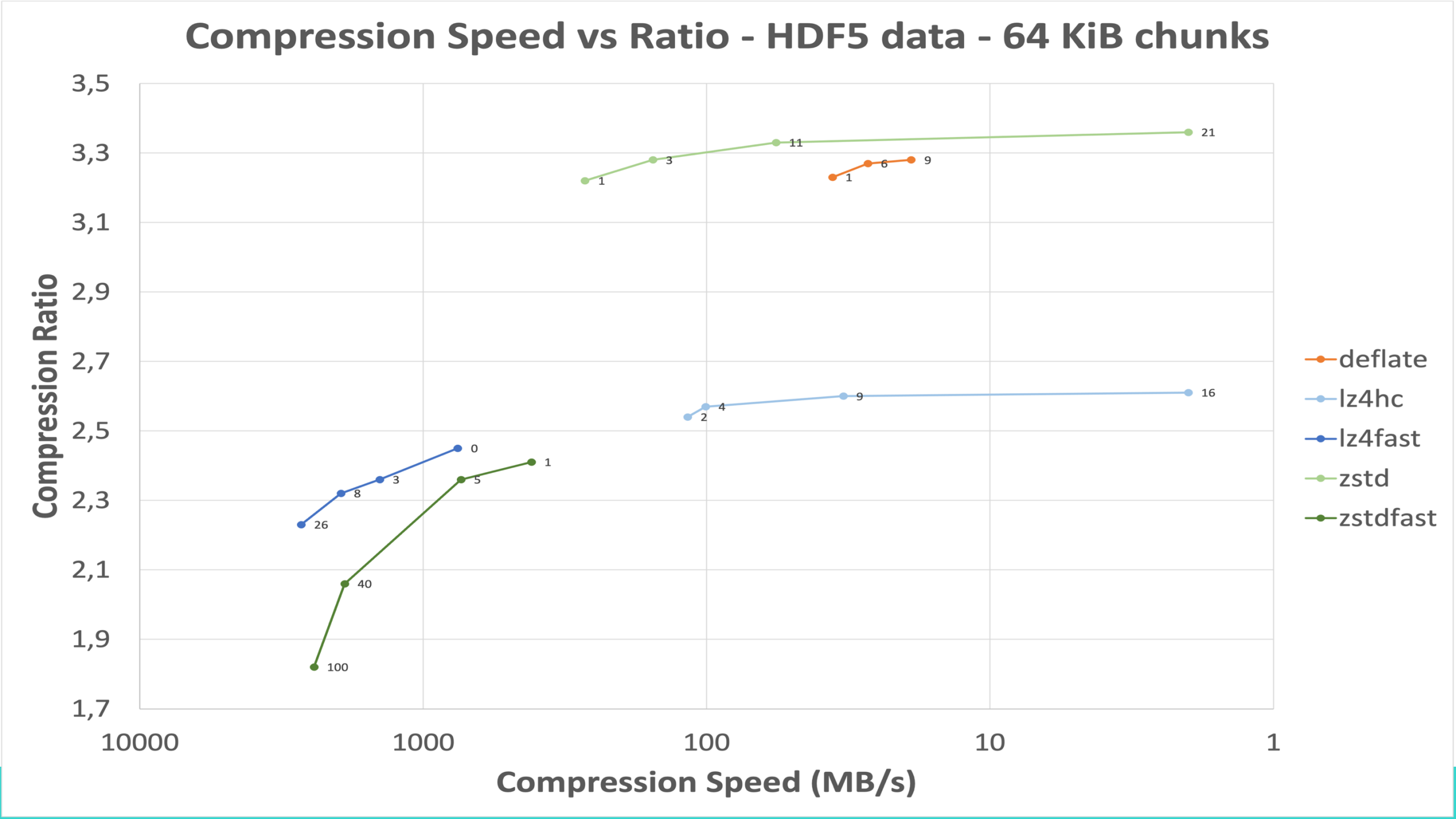
Incompressible data heuristics in operation (cont.)

Whamcloud

```
# lctl set_param osc.*.stats_compr=0
# write compressiable data
# cat lustre/tests/AMSR_E_L3_DailyOcean_V05_20111003.hdf >> /mnt/lustre/foofile
# lctl get_param osc.lustre*.stats_compr

$ sudo lctl get_param osc.lustre*.stats_compr
osc.lustre-OST0000-osc-ffff565948a4a000.stats_compr=
write_pages_compr: 2080
write_pages_uncompr: 1492
write_chunks_compr: 130
write_chunks_no_compr: 93
write_bytes_compr: 3665644
write_bytes_raw: 14629144
write_bytes_incompr: 6111232
read_pages_compr: 0
read_chunks_compr: 0
read_bytes_compr: 0
read_bytes_raw: 0
```

```
$ du -h /mnt/lustre/foofile
13M /mnt/lustre/foofile
```


Compression Speed vs Ratio - HDF5 data - 64 KiB chunks
Compression Ratio
3,5
3,3
3,1
2,9
2,7
2,5
2,3
2,1
1,9
1,7
10000
1000
100
10
1
Compression Speed (MB/s)
deflate
lz4hc
lz4fast
zstd
zstdfast
1
3
11
21
1
6
9
2
4
9
16
0
3
8
26
1
5
40
100

Things for consideration

Whamcloud

- All DIO operations with compressed files will be switched to buffer mode to compress data
- CSDC uses sparse files to store data on the OST. It is not useful to preallocate space for them
  - Depending on client parameter either disable **fallocate** (default) or disable compression
- No sparse file read support in LNet, so no reduction of read **network traffic** (will be fixed)
- No **recompression** of data on read-modify-write to avoid overhead
  - If file updated with small writes, updated chunks become uncompressed
  - Can migrate file to recompress afterward
- No **DoM** support yet (will be fixed)
- No **encryption** support yet (will be fixed)
- **HDD** not recommended (performance is poor due to seeking)
- **T10PI** not integrated yet (server decompression/rewrite changes RPC checksum)
- **GPU Direct** not possible with compression since CPU cannot access data pages in GPU RAM.
  - This is the same with encryption, and GDS must also disabled for those files

Conclusion

Whamcloud

- Compress data that **can be compressed**
- Write **full chunks**
- Adjust **compressibility heuristics** parameters if needed
- Use **buffered** operations
- Use **flash-based** OST storage
- Consider which **other Lustre FS features are** used with CSDC
- Use compression during **migration**
- Use **progressive layout** to enable compression for large files
- Balance client CPU usage by using different compression **algorithms** and **levels**

Whamcloud

Thank You!
Questions?