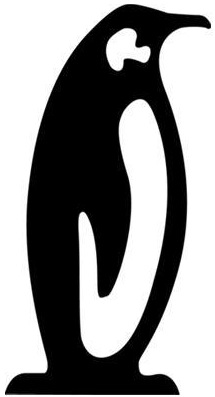


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

TÉCNICO LISBOA

UNIVERSITY OF OXFORD

Optimizing Checkpoints with Built-in Memory Page Compression

Radostin Stoyanov – PhD Student @ Scientific Computing Group
Adrian Reber – Senior Principal Software Engineer
Prof. Rodrigo Bruno, Prof. Wes Armour

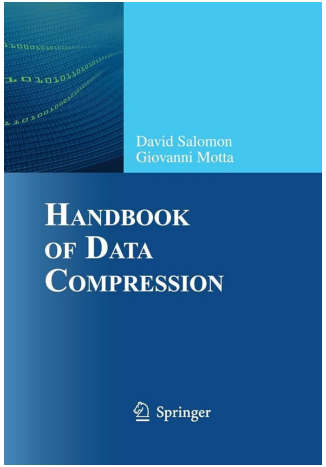
東京 2025 LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Red Hat

Background - Data Compression

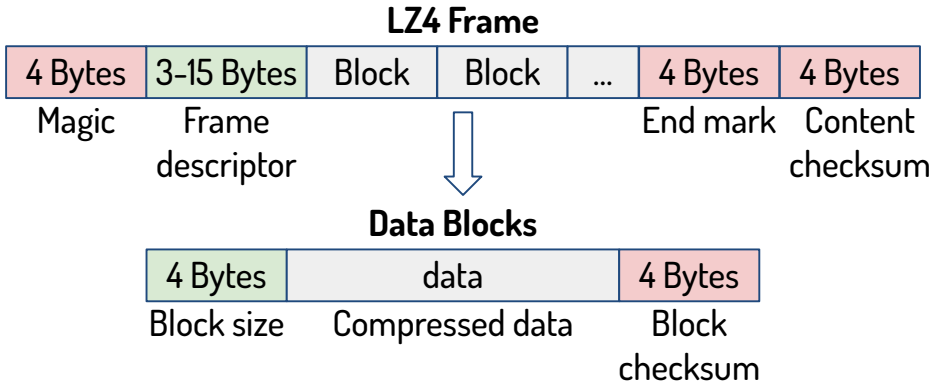
- **LZ (Lempel-Ziv) Family**
  - LZ4, LZO, zstd, libdeflate, snappy, etc.
  - Very fast with good compression ratio
  - Suitable for live migration and frequent checkpoints

- **LZMA/LZMA2 (Lempel-Ziv-Markov chain Algorithm)**
  - xz-utils, liblzma, lzma-sdk, p7zip, etc.
  - Prioritize compression ratio over speed
  - Suitable for checkpoint archival and long-term retention



Handbook of Data Compression
By David Salomon and Giovanni Motta.

Background - LZ4 Overview



Why LZ4?

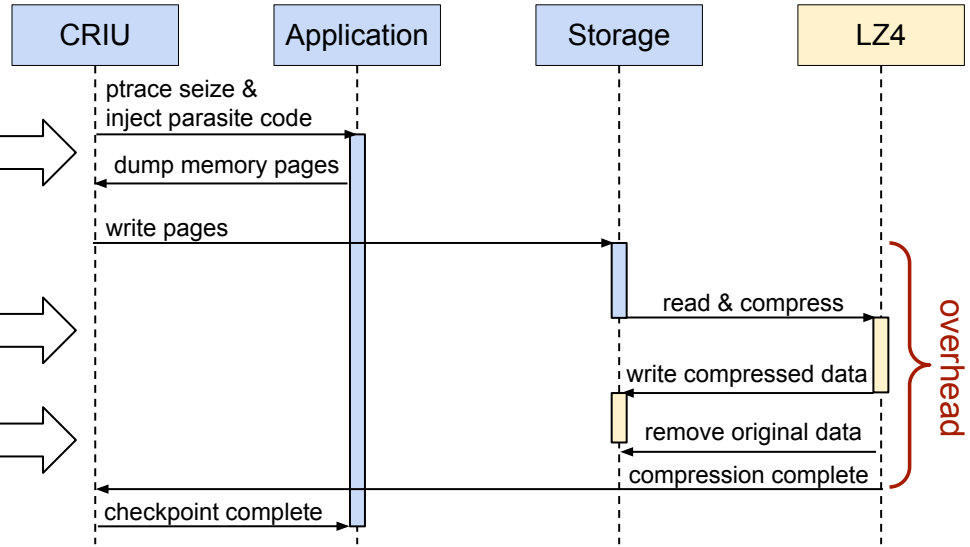
- Extremely fast (de)compression
- Good compression ratio
- Small memory footprint

Naïve Compression & Decompression

Example: OpenJDK Coordinated Restore at Checkpoint (CRaC)
https://github.com/CRaC/criu

Naïve Checkpoint Compression

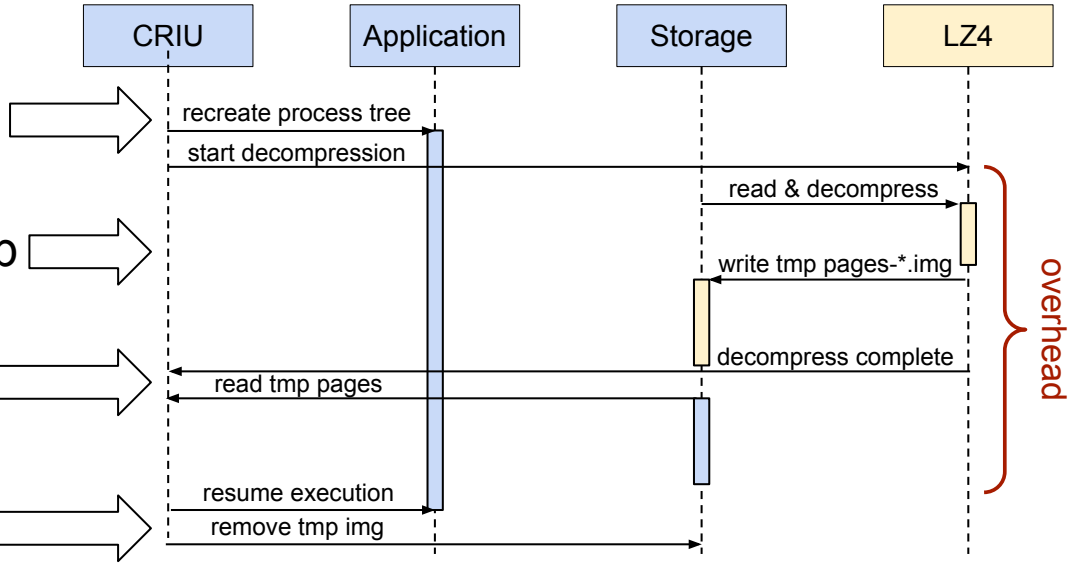
1. Create Checkpoint
2. Compress Image Files
3. Remove Original Images



OpenJDK CRaC: https://github.com/CRaC/criu

Naïve Restore with Decompression

1. Load Checkpoint Images
2. Decompress Pages in /tmp
3. Restore from Temp Files
4. Remove Temp Files



OpenJDK CRaC: https://github.com/CRaC/criu

Challenges with Naïve Approach

- Inefficient CPU utilization
  - High rate of L1-L3 cache misses & Low instructions-per-cycle
- Not suitable for large checkpoints
  - **Required storage = Uncompressed pages + Compressed pages**
- **Incremental Checkpoints** and **Deduplication** are not supported
  - Pre-copy (iterative) migration: `criu pre-dump …` ----▸ `criu dump …`
  - Incremental checkpoints: `criu dump --track-mem --leave-running …`
  - Restore from in-memory filesystem: `criu restore --auto-dedup …`

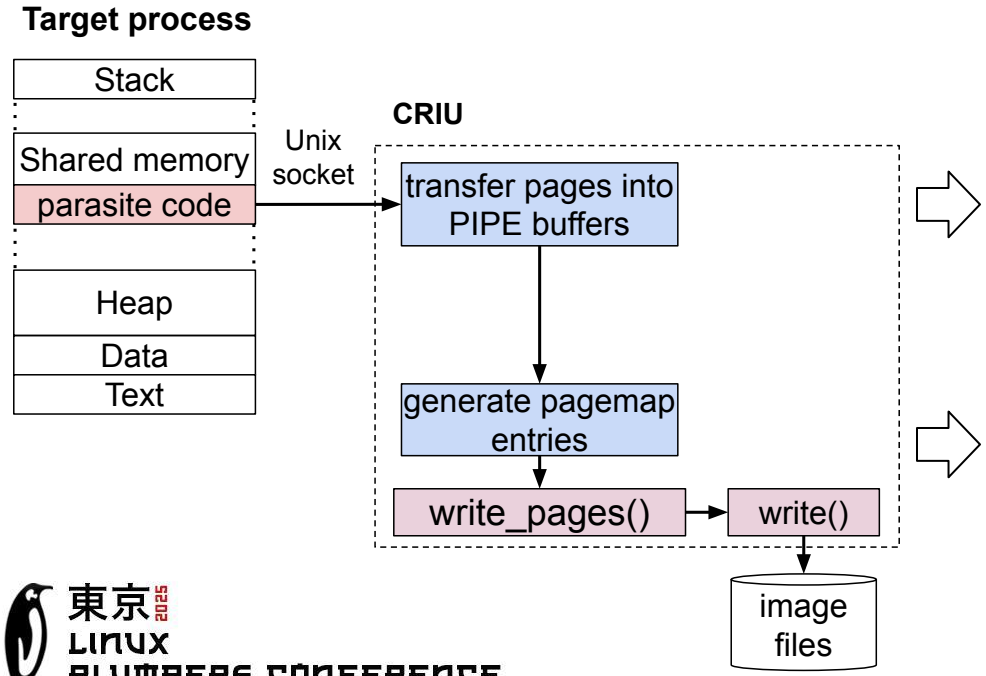
CRIU with Built-in Pages Compression

Towards Efficient End-to-End Encryption for Container Checkpointing Systems
ACM SIGOPS Asia-Pacific Workshop on Systems. 2024.

東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Memory Checkpointing

Pagemap indicates the **file offset** and **length** to read from pages image



pagemap-<PID>.img

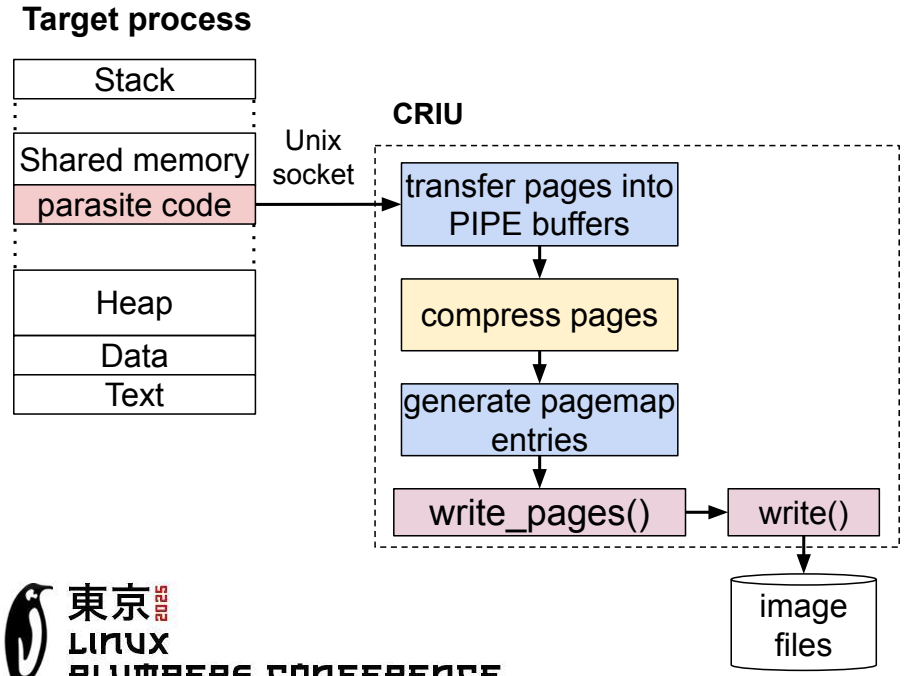
```
...
{ "vaddr": "0x55d24a9fe000", "nr_pages": 2 },
{ "vaddr": "0x55d24aa06000", "nr_pages": 1 },
{ "vaddr": "0x7f03acc00000", "nr_pages": 1021 },
{ "vaddr": "0x7f03acffd000", "nr_pages": 1024 },
...
```

pages-<ID>.img

```
...
00007b0 8349 01c5 8041 007d 742d 48f5 3d8b 285e
00007c0 0000 8548 0fff 0584 ffff 31ff 4cd2 358d
00007d0 284c 0000 5589 ebb8 0f1c 801f 0000 0000
00007e0 8b49 107e 8349 10c6 4583 01b8 8548 0fff
00007f0 db84 fffe 4cff ee89 73e8 fffd 85ff 75c0
0000800 4cdf 6d63 48b8 0d8d 2814 0000 894c 48e8
0000810 e0c1 4804 c801 8348 0038 840f feb0 ffff
0000820 8b49 187f 8548 0fff 3b84 0002 4900 e5c1
0000830 4804 058d 27e8 0000 8b46 286c 4108 fd83
...
```

CRIU with Built-in Compression

Compressing **independent pages** and storing **compressed size**



inventory.proto

```
message inventory_entry {
...
+    optional bool pages_compression = 15;
}
```

pagemap.proto

```
message pagemap_entry {
        required uint64 vaddr                   = 1;
        required uint32 compat_nr_pages         = 2;
        optional bool   in_parent               = 3;
        optional uint32 flags                   = 4;
        optional uint64 nr_pages                = 5;
+       repeated uint32 compressed_size          = 6;
+       optional uint32 total_compressed_size   = 7;
}
```

Restoring Compressed Memory

Extending **page-read engine** with support for compressed pages

```
/* Engine reading pages from image file(s) */
struct page_read {
  ...
  /* remote, img_streamer, local or compressed */
  int (*maybe_read_page)(...);

  /* Index of the current compressed page size */
  int compressed_size_index;
  ...
};
```

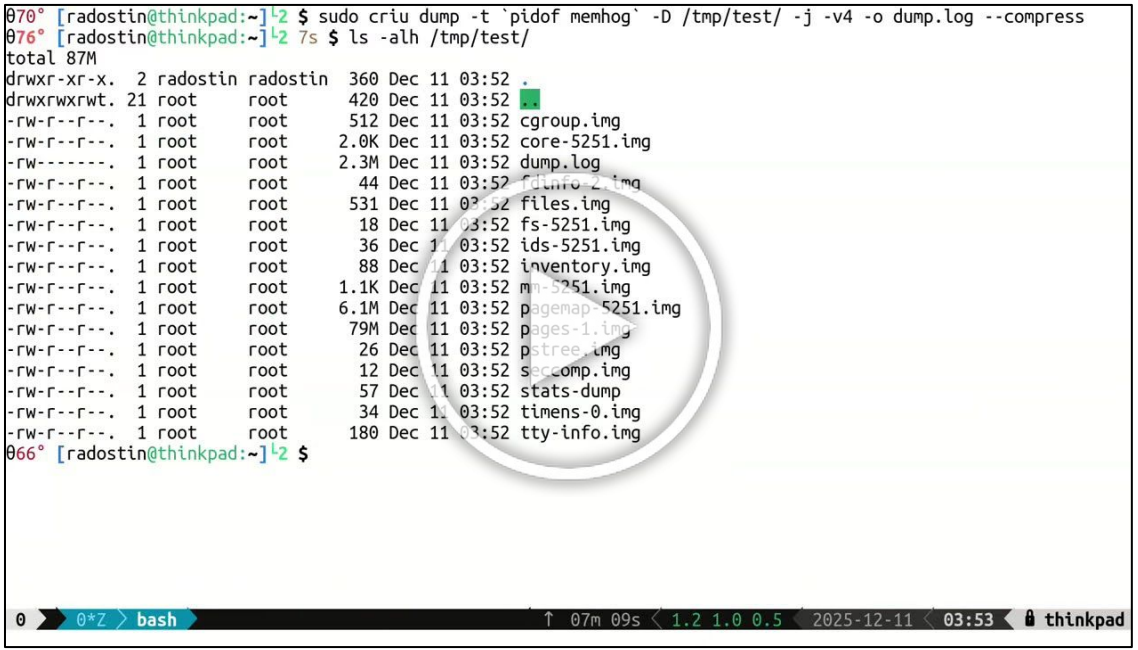
```
/* One "job" for the preadv() syscall */
struct page_read_iov {
  ...
  size_t n_compressed_size;
  uint32_t *compressed_size;
  uint32_t total_compressed_size;
  ...
};
```

Deduplicating Compressed Pages

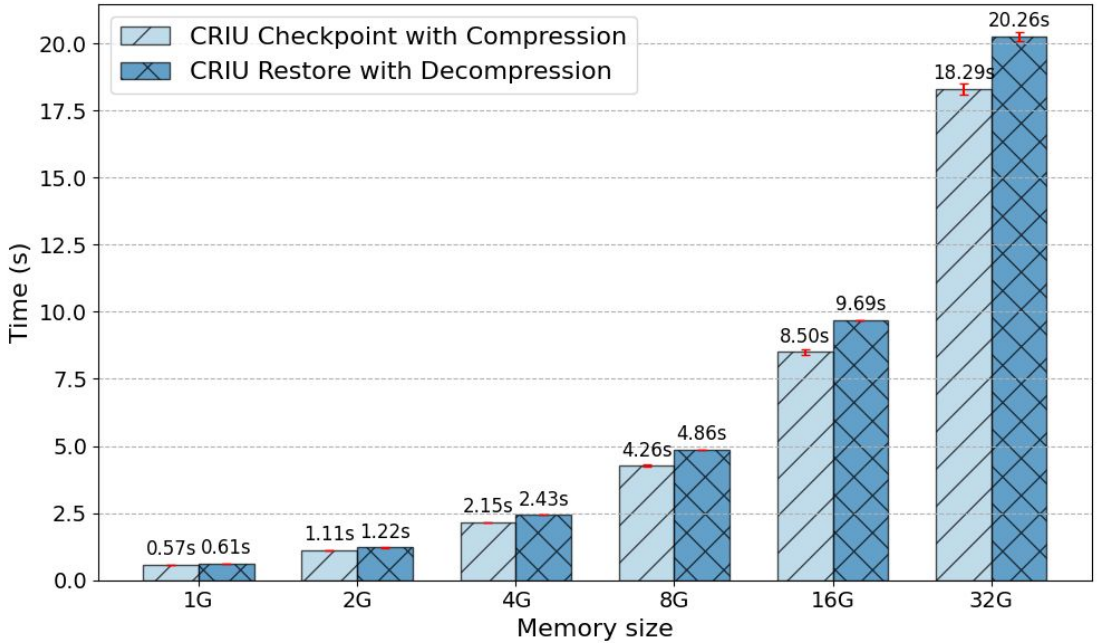
Compute file offset for `fallocate(PUNCH_HOLE)` using compressed size of pages

```
size_t i = pr->compressed_size_index, len = 0;

for (int j = 0; j < nr; j++) {
    size_t csize = pr->pe->compressed_size[i+j];

    pread(fd, buf, csize, pr->pe->pi_off + len);
    decompress_data(...);
    len += csize;
}

punch_hole(pr, pr->pi_off, len);

pr->pi_off += len;
pr->compressed_size_index += nr;
```

CRIU Built-in Compression Demo



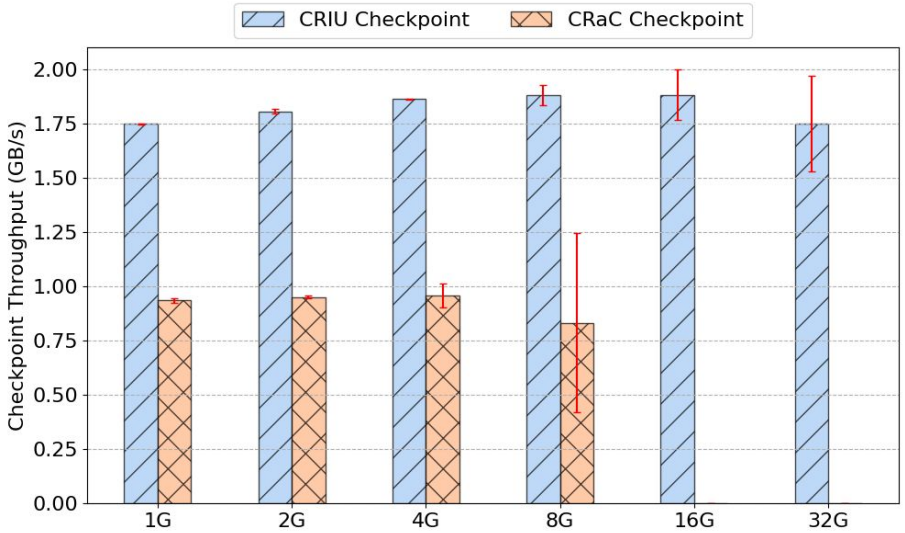
Preliminary Evaluation

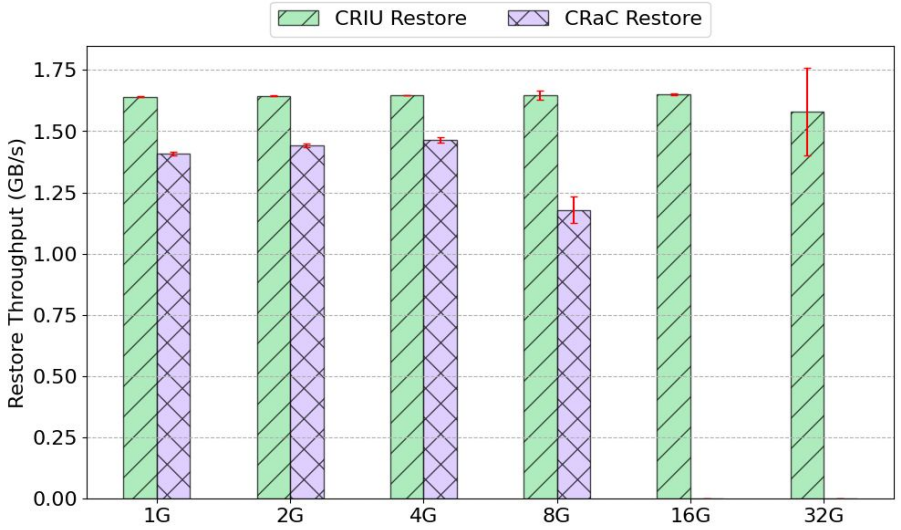


memhog size	compressed pages
1 GB	6.6 MB
2 GB	14 MB
4 GB	27 MB
8 GB	53 MB
16 GB	105 MB
32 GB	209 MB

memhog repeatedly calls `memset()` to fill memory with `0xff`, creating a highly compressible repeated pattern.

Preliminary Evaluation





Naïve approach (CRaC) fails with out-of-memory error for memory allocations >=16GB.

Summary & Questions

- Built-in compression for memory pages
- Reduced performance overhead & storage requirements
- Seamless integration with container runtimes and Kubernetes