Exploring Userspace Memory Mapping for RDMA-Enabled Network-Attached Memory

Jacob Wahlgren
Department of Computer Science, KTH Royal Institute of Technology Stockholm, Sweden

Jennifer Faj
Department of Computer Science, KTH Royal Institute of Technology Stockholm, Sweden

Eric Green
Lawrence Livermore National Laboratory, USA

Maya Gokhale
Lawrence Livermore National Laboratory, USA

Ivy Peng
Department of Computer Science, KTH Royal Institute of Technology Stockholm, Sweden

Motivation

- Memory-bound applications such as graph processing applications often require large memory capacity that exceeds a node's physical memory (Fig. 1).
- Currently, high-performance computing (HPC) systems provide massive amount of compute nodes and use resource over-provisioning to support a diverse set of workloads.
- Network-attached memory can be used to back an application's virtual memory space when the local compute node has exhausted its physical memory, enabling compute and memory disaggregation.

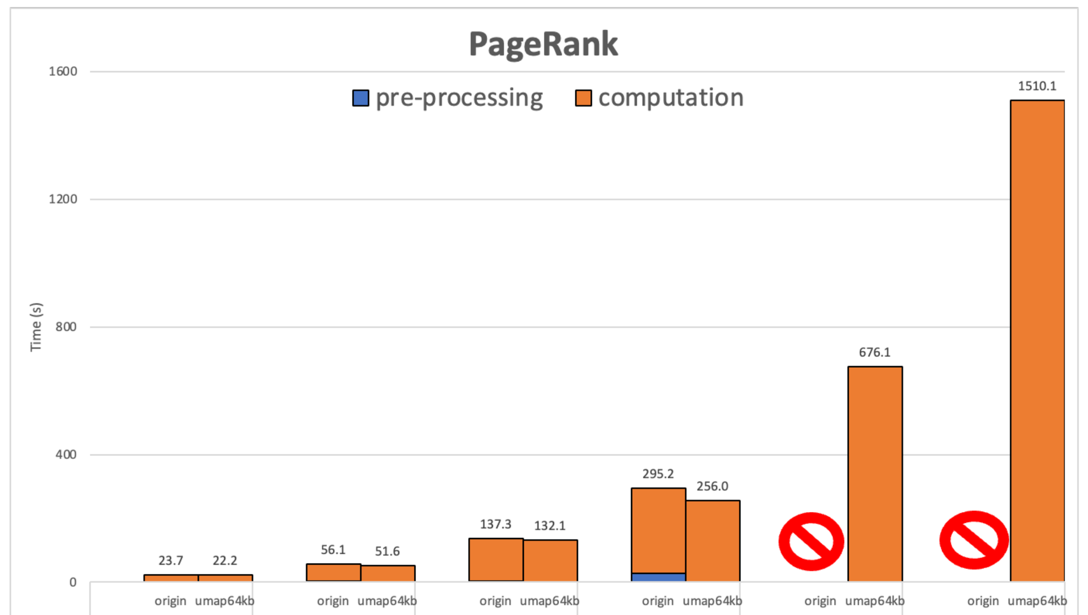


Fig 1: Ligra graph processing framework performs in-memory graph processing. It is bound by memory capacity of a node because it allocates large intermediate data structures for graph pre-processing

Background:

- Umap[1] is a library that enables user-space paging management through *userfaultfd*
- Datastores are supported on different tiers of memory and storage hierarchy, e.g., NVMe SSD
- Support application-specific configurations of page size, concurrency control, buffer size, and prefetching policies
- RPC and MPI were explored previously for enabling network-attached memory

Design

- Memory regions are registered to offload page fault handing into user space and remote memory regions are created and mapped (Fig. 2)
- Internal manager schedules *fetch* requests to *Fillers* and *evict* requests to *Evictors*
- Workers create work request to one-sided read and write from memory regions over the network
- Page-level lossless and lossy data compression for reducing data movement

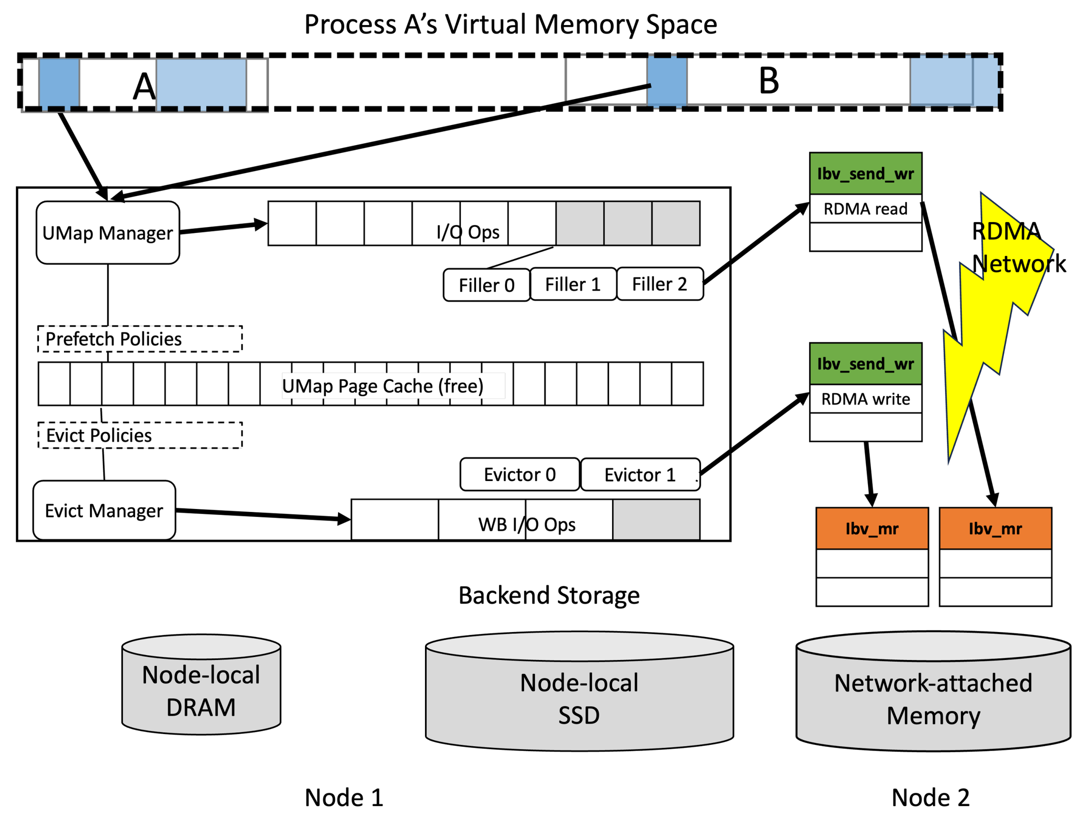


Fig 2: Overview Architecture

Implementation in C++/C:

userfaultfd: page faults handling in user space

libibverbs: RDMA verbs in user space

LZ4: Fast lossless compression library

ZFP[3]: Lossy compression of floating-point data

An example code snippet for allocating and accessing an array on network-attached memory

```
# include "umap/umap.h"

//create a network-attached datastore
Umap::Store* store_a = new Umap::StoreNetwork("a", length, network_client,
                                              compression_mode);

//create a memory mapping to the datastore
double* a = (double*) umap_ex(NULL, length, PROT_READ|PROT_WRITE,
                              UMAP_PRIVATE, 0, 0, store_a);

//access the memory region as if in main memory
for (size_t j=0; j<array_size; j++) {
    a[j] = 1.0;
}
```

Preliminary Results

Testbeds: dual-socket AMD EPYC 7401 at Livermore computing, Bluefield-2 NIC interconnected with 2x53.125 Gbps link through an IB switch. The peak BW by linux-rdma/perftest is 11700MB/s.

Benchmark: extended from the original STREAM with allocation of main data objects in the network-attached remote memory regions

Finding I: The concurrency level of both Filler and Evictor workers have a high impact on the performance. Userspace control is important for performance tuning.

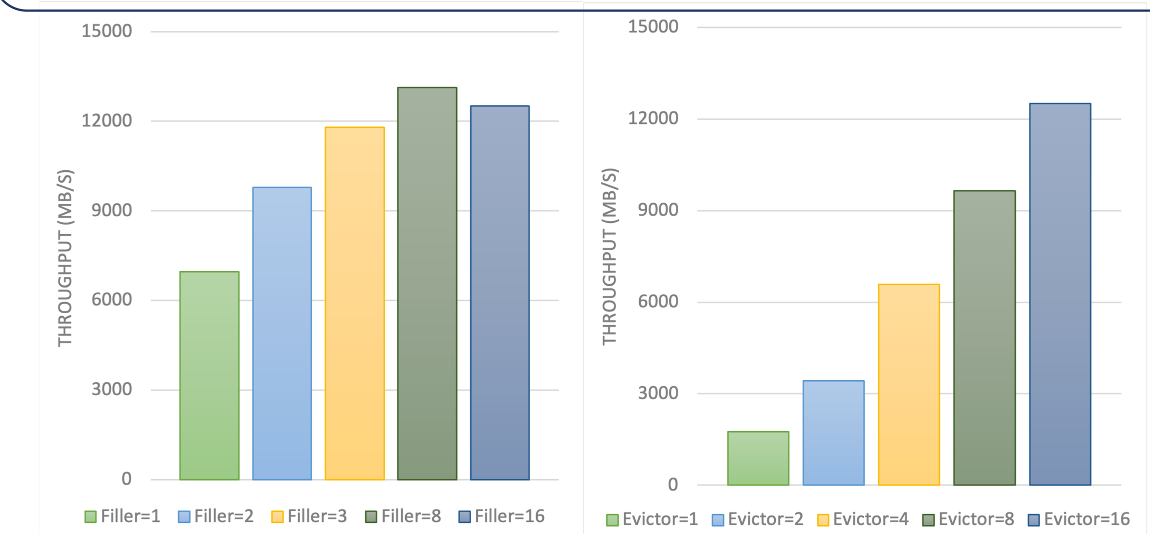


Fig 3: The performance obtained at an increased number of filler and evictor workers

Finding II: Using data compression at page level could improve the overall throughput (~50% in Fig. 4) when the reduced network traffic outweighs the additional compression and decompression overhead.

Fig 4: Throughput Improvement using LZ4 compression

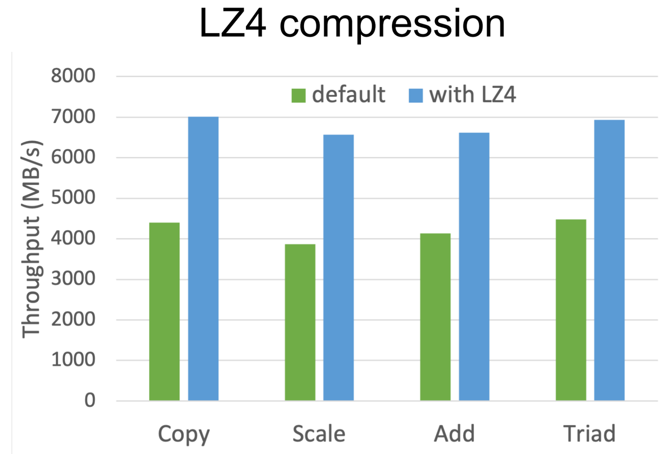
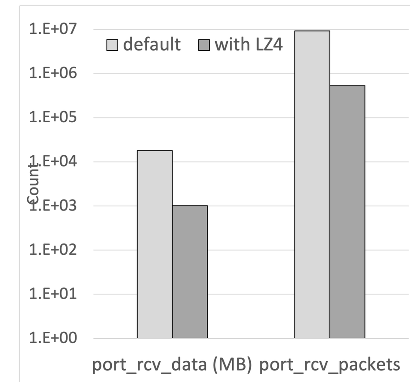
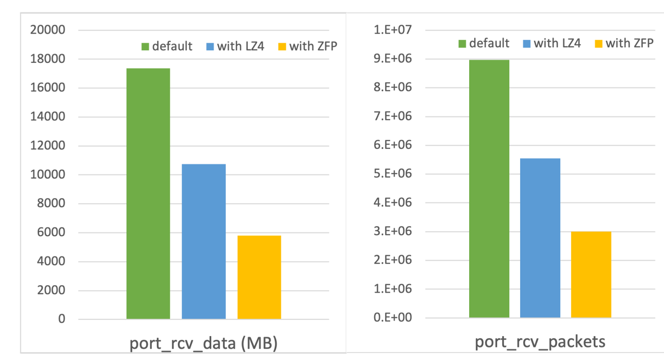


Fig 5: Network Traffic with and without LZ4



Finding III: Lossy compression may be more effective for reducing data movement over network than lossless compression for some floating-point values

Fig 6: Throughput with varying numbers of Filler and Evictor workers



Finding IV: some floating-point datasets could even have higher compressed data size by *LZ4* than their original size

Future Works

Recent Nvidia BlueField DPU provides hardware acceleration of data compression. We compare the compression ratio and time using (1) DPU deflate hardware unit (2) LZ4 on DPU's ARM core (3) LZ4 on Host side, on a set of real scientific datasets from 12 scientific simulations[2]

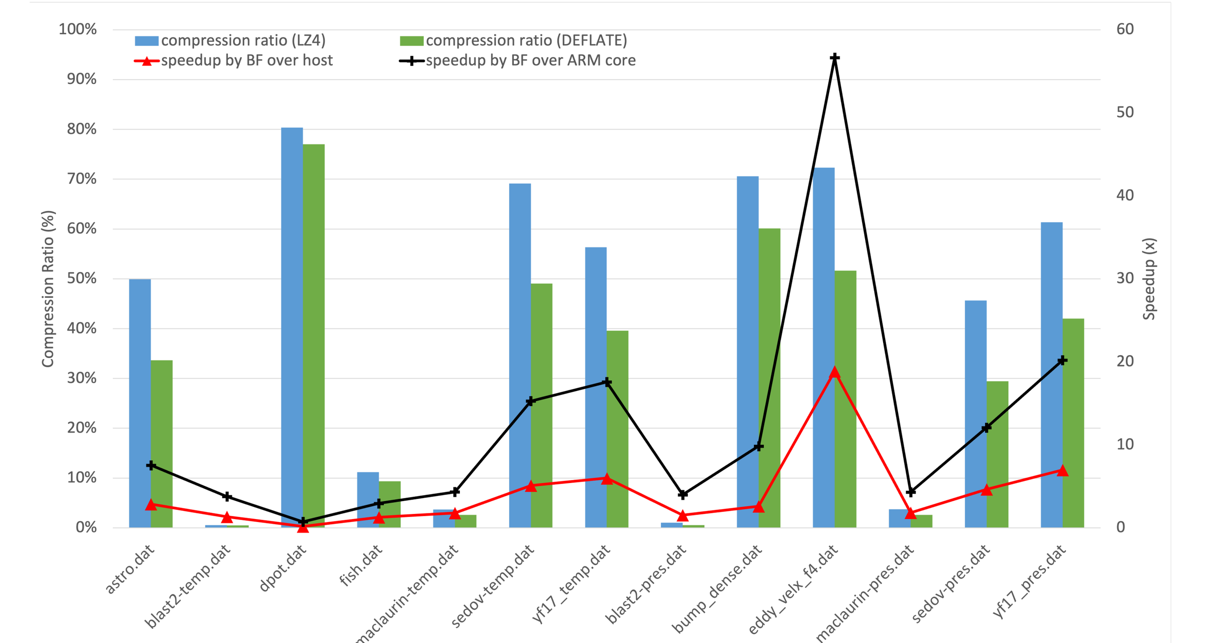


Fig 7: Compression ratio and performance of DEFLATE and LZ4 on ARM and AMD hosts

Summary:

In this work, we extend a userspace paging management library to enable memory mapping RDMA-enabled memory regions over network. We show that an optimal userspace control of worker threads achieves up to 70% improvement. Using LZ4 data compression could significantly reduce data movement while sustaining performance, but in some cases may reduce performance. Future work will explore online adaptation of compression mode and hardware offloading.

References

[1] Peng, Ivy, et al. "UMap: Enabling application-driven optimizations for page management." 2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC).
[2] Lu, Tao, et al. "Understanding and modeling lossy compression schemes on HPC scientific data." 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2018.
[3] Lindstrom, Peter. "Fixed-rate compressed floating-point arrays." IEEE transactions on visualization and computer graphics 20.12 (2014): 2674-2683.

Acknowledgement

This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344 with support from the DOE Exascale Computing Project. LLNL-POST-854442.

Lawrence Livermore National Laboratory

GitHub