Exploring Userspace Memory Mapping for RDMA-enabled Network-attached Memory

Jacob Wahlgren, Jennifer Faj, Ivy Peng
KTH Royal Institute of Technology
Sweden
{jacobwah,faj,ivybopeng}@kth.se

Eric Green, Maya Gokhale
Lawrence Livermore National Laboratory
USA
{green77,gokhale2}@llnl.gov

Abstract

Memory-bound applications like graph processing applications often require large memory capacity beyond a single node. Current HPC systems overprovision compute and memory resources to meet requirements of diverse workloads. In this work, we explore using network-attached memory for disaggregating memory from compute nodes to satisfy the demand of memory-intensive workloads. We provide a library that enables applications to access network-attached memory as if in its main memory, and exposes critical controls to userspace, including concurrency level and page-level data compression. Our preliminary results show that the flexibility of tuning concurrency and compression is important for improving performance and reducing data movement. Also, our results on 12 scientific data sets indicate that DPU compression offloading could significantly speed up compression and is important for future optimizations.

Keywords: RDMA, network-attached memory, memory mapping

Introduction. The memory capacity of a node determines the feasibility of running large-scale applications, especially for memory-bound applications with high memory footprints. As today's HPC systems employ tightly coupled compute and memory resources in each node, users must request enough nodes to fulfill the memory requirement, which could leave the computing resources underutilized. For instance, Ligra[5] is a popular in-memory graph processing framework that allocates large intermediate data structures during graph preprocessing, severely limiting the scale of problems feasible on an HPC system. Network-attached memory is a form of disaggregated memory where a process's virtual memory is backed (partially) with the physical memory of another node. Many previous works, e.g., infiniswap [1], have proposed kernel-based solutions for leveraging network-attached memory. Instead, in this work, we explore a userspace paging library that enables applications to access network-attached memory as if in its main memory. In particular, critical controls, such as concurrency level and page-level data compression, are exposed to userspace for flexible application-specific tuning.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344. LLNL-ABS-854524.

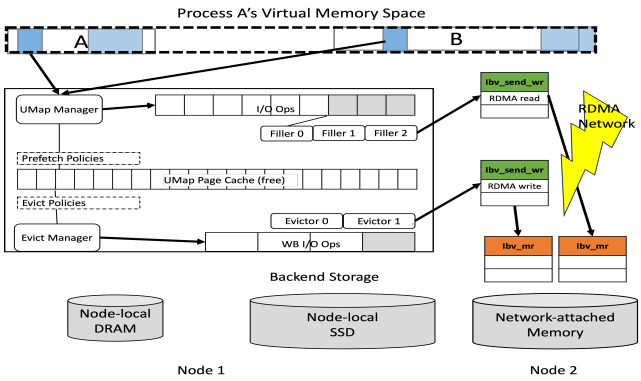

Figure 1. Overall architecture of RDMA-enabled backend storage for extending a process's memory space on Node 1.

Design and Implementation. We extend a user-space paging fault handler library called UMap [4] to support network-attached remote memory regions (MRs) with RDMA-enabled data transfer. Fig. 1 illustrates the overall architecture, where an application process A runs on Node 1 while the physical memory on Node 2 is used to back parts of process A's virtual memory address space, e.g., A and B. The library provides simple APIs for an application to create *network-attached datastores* and then map them into its virtual address space so that an application can access it at byte granularity. Internally, the library manages RDMA-enabled memory regions and their mapping to application's virtual memory address. Leveraging the fact that a single page cannot be paged in and out concurrently, two separate completion queues are managed internally for enhanced parallelism. The datastores are managed in equal-sized *pages* whose size can be controlled in userspace. When a page is accessed, a *Filler* thread creates a work request to read its mapped remote MR and then caches it in Node 1's main memory. When free space in the page cache is low (the size is controlled in userspace), *Evictor* threads evict old pages and write dirty pages back to its mapped remote MR. The user can control which compression mode to enable on pages to reduce data movement on network. LZ4 is used for lossless compression, and ZFP[2] for lossy compression. When compression is enabled, a Evictor will compress a page before sending over network and record its compressed size in a global metadata, and a Filler will decompress a page if it is marked as compressed.

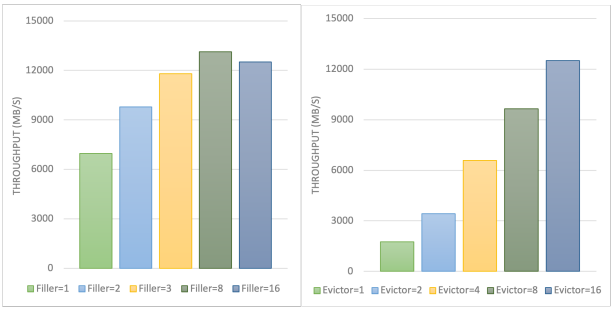

Figure 2. The measured throughput from network-stream copy kernel obtained using different numbers of Filler (left) and Evictor (right) workers.

Figure 3. The measured throughput (left) and data movement over network (right) using the default implementation and the implementation with LZ4 data compression.

Preliminary Results. We evaluated the library on a cluster at Livermore Computing, with two AMD EPYC 7401 CPUs and one Bluefield-2 NIC per node, and 100 Gbps Ethernet. The peak bandwidth measured by linux-rdma/perftest is 11700MB/s. We extended the STREAM benchmark to allocate main data structures in network-attached remote memory regions. We configured the page cache size to have 40% of memory footprint offloaded to network-attached memory.

Fig. 2 presents a scaling test of the number of Filler and Evictor threads, respectively. The results show that the two types of workers have a high impact on performance. The performance is more sensitive to the number of evictors as the eviction task has higher latency. With eight fillers and 16 evictors, the throughput closely approximates the peak bandwidth of the network. The results show that userspace configuration is critical for performance optimization.

We evaluate the effect of data compression by enabling LZ4 compression. Fig. 3 shows that enabling the compression increases the throughput of four kernels by 50% - 70%. We quantify the data movement using Linux's rdma-statistic counters. Note that the y-axis in the right panel of Fig. 3 is in logarithmic scale, and it shows that data movement is reduced by an order. However, as a lossless compression, we also found that LZ4 may result in a compressed size higher than the original, depending on the data values. Switching to lossy compression could be beneficial for such cases. Our current implementation disables compression when such a scenario is detected, while an online adaptive scheme can be designed for future optimization.

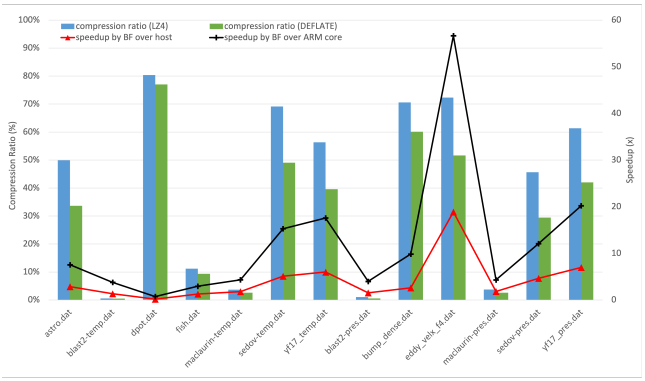

Figure 4. Compression ratio and performance using BlueField compression hardware unit (DEFLATE) and software compression (LZ4) on ARM and AMD Host

Recent programmable NICs like Nvidia's Bluefield DPU provide a hardware unit for compression. To evaluate the potential of offloading data compression onto the network, we used a set of real scientific datasets [3] and evaluated the compression ratio and time using DPU's hardware unit (DEFLATE), DPU's general ARM core, and the AMD host core. Fig. 4 shows that DPU's compression ratio (green) is always lower than the software compression (blue), indicating further reduced data movement. On average, DPU's hardware compression achieves 4× speedup of the AMD host and 12× of DPU's ARM core.

Conclusions and Future Work. In this work, we extend a userspace paging management library to enable memory mapping RDMA-enabled memory regions over network. Our results show that adapting concurrency improves throughput by 4×, and enabling compression reduces data movement by an order. For future work, we evaluate hardware-based compression offloading to DPU using real scientific datasets and highlight the potential of significant performance gain.

References

[1] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. 2017. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 649–667.

[2] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2674–2683.

[3] Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Jong Choi, Norbert Podhorszki, Scott Klasky, Mathew Wolf, Tong Liu, et al. 2018. Understanding and modeling lossy compression schemes on HPC scientific data. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 348–357.

[4] Ivy Peng, Marty McFadden, Eric Green, Keita Iwabuchi, Kai Wu, Dong Li, Roger Pearce, and Maya Gokhale. 2019. UMap: Enabling application-driven optimizations for page management. In *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE, 71–78.

[5] Julian Shun and Guy E Blelloch. 2013. Ligra: a lightweight graph processing framework for shared memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 135–146.